Advanced Programming On the Atari ST

COMPUTE!

$3.00
July
1986
Issue 74
Vol. 8, No. 7
$3.95 Canada
02193
ISSN 0194-357X

The Leading Magazine Of Home, Educational, And Recreational Computing

Screen Handler For Commodore 64 & 128
Enhanced Input For BASIC Programs

Hex War
Strategic Simulation For Commodore 64, 128, Amiga, Atari, Apple, And IBM PC/PCjr

Screen Machine II
Advanced Drawing Program For IBM PC/PCjr

Sound Development System For Atari 400/800, XL, XE

User's Poll:
The Top Five FREE Programs For Your Computer

C And The 68000
An Overview Of This Popular Language

Apple Catalog Sorter
For ProDOS Disks

HOW PEOPLE WITH COMMON INTERESTS FIND AN INTERESTING COMMON GROUND.

Presenting CompuServe Forums. Where people from all over get together, without even leaving home.

Now thanks to CompuServe Forums, computer owners are sharing common interests by talking to each other through their computer keyboards. ***Software users, computer enthusiasts, ham operators, french cooks, fire fighters, science fiction lovers and other special interest groups*** are already in touch, online.

Because when you subscribe to CompuServe, you're able to reach people who want to talk about the things you do. As many people as you like. For as long as you like. Whenever you wish.

Join a conversation already in progress or start one on your own. Ask questions. And get answers.

All it takes is a modem, most any personal computer and CompuServe.

Forum members across the country are as close as a local phone call.

You can go online with just a local call in most major metropolitan areas. And normal usage fees for weekday nights and weekends are just 10¢ a minute.

Of special interest to all Forum participants is software that's FREE for the taking.

Public domain software. For all sorts of activities, from games to business programs. And it's just as easy to copy a piece of software as it is to participate in a Forum.

Become a CompuServe subscriber and get a $25 Usage Credit to start you off.

Becoming a subscriber is as easy as contacting your local computer dealer. Or you can call us and order direct. Suggested retail price is $39.95.

And if you'd want more information about CompuServe, we'll be happy to send you a free brochure. Because with all that CompuServe offers—we think it's in your best interest.

CompuServe®

Information Services, P.O. Box 20212, 5000 Arlington Centre Blvd. Columbus, OH 43220
800-848-8199
In Ohio, call 614 457-0802

An H&R Block Company

$10,000.00
Atari ST Programming Contest!

First Prize $5,000.00 **Second Prize $2,500.00**

Third Prize $1,000.00

Three Honorable Mentions $500.00 each

COMPUTE! Publications, Inc. is looking for the very best original software for the Atari ST series computers. And to prove we're serious, we're offering a total of $10,000.00 in prize money to the top six winners. That's $5,000.00 for First Prize, $2,500.00 for Second Prize, $1,000.00 for Third Prize, and $500.00 each for three Honorable Mentions. In addition, the winners will receive our standard royalties when their programs are published. And even if your program doesn't win a prize, you can still earn purchase fees and royalties if we accept your entry for publication.

Interested? If so, read these rules:

1. Entries must be your original work, previously unpublished. All those whose programs are accepted will be required to affirm this in writing.

2. You can submit as many entries as you want, but we cannot consider programs which have been entered in other contests or submitted for publication elsewhere at the same time.

3. The deadline is October 1, 1986. All entries must be received at our offices by this date. Programs submitted after this date will still be considered for publication, but will not be entered in the contest.

4. Entries are allowed (and encouraged) in virtually all software categories: home and business applications, education, recreation, telecommunications, graphics, sound and music, utilities, and desk accessories.

5. Entries may be written in any programming language—including BASIC, Logo, C, machine language, Pascal, Modula-2, Forth, FORTRAN, and Prolog—as long as they meet two requirements. First, if you're using a compiled language, the compiled object or run-time code must be a self-standing program that can be run by someone who doesn't own a copy of the language. (Exceptions are ST BASIC and Logo. Since these languages come with the ST, it can be assumed that everyone owns a copy.) Second, we must be able to legally distribute the program without incurring licensing fees or other obligations to the maker of the language. If you're not sure whether a certain language qualifies, contact its maker for clarification.

6. Entries must be submitted on a single- or double-sided 3½-inch ST disk with both the run-time code and source code included.

7. Entries must be accompanied by an article which explains how to use the program, what it does, and so on. If your program employs any new or unusual techniques that you think will be of interest to other ST programmers, you can also describe how the program works.

8. Submissions which do not win a prize and are not accepted for publication will be returned only if accompanied by a self-addressed, stamped mailer.

9. All judging will be handled by the staff of COMPUTE! Publications, Inc. All decisions regarding contest entries and acceptances will be solely at the discretion of COMPUTE! Publications, Inc., and all decisions are final. This includes decisions regarding creativity, similarity among entries, and so forth.

10. Winners will be announced by COMPUTE! Publications, Inc. in late 1986.

11. This contest is void where prohibited by law. Full-time, part-time & previous employees of COMPUTE! Publications, Inc., and Capital Cities/American Broadcasting Corporation are ineligible for the contest, but may still submit work for publication at standard rates.

Every Contest Entry Must Contain This Form:

I warrant that the program presently entitled____ is my own original work and that the work has not been submitted for consideration elsewhere, nor has it been previously published. If my work is accepted by you, I understand that your decision as to the selection of winners and awarding of prizes is final and without recourse on my part. I agree, should you select my submission, to sign your standard contract, which includes assignment of the copyright of the program to COMPUTE!, and to allow you to use my name and image in promotional materials and other forms. *(If you are under age eighteen, your parent or legal guardian must sign for you.)*

Address entries to:
ATARI ST CONTEST
COMPUTE! Publications, Inc.
P.O. Box 5406
Greensboro, NC 27403

COMPUTE! Publications, Inc. abc
Part of ABC Consumer Magazines, Inc.
One of the ABC Publishing Companies

COMPUTE!

JUNE 1986
VOLUME 8
NUMBER 7
ISSUE 74

GUIDE TO ARTICLES AND PROGRAMS

FEATURES

REVIEWS

COLUMNS AND DEPARTMENTS

THE JOURNAL

NOTE: See page 121 before typing in programs.

AP Apple, **Mac** Macintosh, **AT** Atari, **ST** Atari ST, **V** VIC-20, **64** Commodore 64, **+4** Commodore Plus/4, **16** Commodore 16, **128** Commodore 128, **P** PET/CBM, **TI** Texas Instruments, **PC** IBM PC, **PCjr** IBM PCjr, **AM** Amiga *General interest

COMPUTE! Publications, Inc. abc
Part of ABC Consumer Magazines, Inc.
One of the ABC Publishing Companies
ABC Publishing, President, Robert G. Burton
1330 Avenue of the Americas, New York, New York 10019

COMPUTE! The Journal for Progressive Computing (USPS: 537250) is published monthly by COMPUTE! Publications, Inc., 825 7th Ave., New York, NY 10019 USA. Phone: (212) 265-8360. Editorial Offices are located at 324 West Wendover Avenue, Greensboro, NC 27408. Domestic Subscriptions: 12 issues, $24. POSTMASTER: Send address changes to: **COMPUTE!** Magazine, P.O. Box 10955, Des Moines, IA 50950. Second class postage paid at Greensboro, NC 27403 and additional mailing offices. Entire contents copyright ©1986 by COMPUTE! Publications, Inc. All rights reserved, ISSN 0194-357X

Editor's Notes

Last month, we hinted at a significant pending announcement for Atari ST users. Here at COMPUTE!, one of the most exciting things we do is launch new publications. We are, without parallel, the most successful and balanced publishing house in the industry of consumer computing. A decision on our part to support a computer manufacturer and a computer system with a dedicated magazine is not made lightly. We are extremely pleased, therefore, to announce that ABC Publishing, our parent company, has committed full support to our launch of COMPUTE!'s Atari ST Disk & Magazine.

This will be our very first product that comes as a magazine/disk combination only. Whether you subscribe or purchase it from a newsstand, you'll get a magazine containing the articles and a disk containing the programs. It's a single, united product. And one we're quite proud of.

No publisher in this industry has been as successful as COMPUTE! Publications at marrying diverse publishing technologies. When we introduced COMPUTE!'s GAZETTE DISK, other disk products were selling a few hundred copies at $30 or more per issue. We launched the GAZETTE DISK at $12.95 and created, with your massive support, an overnight price move in the industry. The GAZETTE DISK is today the bestselling product of its kind in the world, circulating tens of thousands of copies per month.

We fully expect COMPUTE!'s Atari ST Disk & Magazine to accomplish the same feat. At launch, our newsstand distribution will rival that of a magazine-only publication. Logistically, there are numerous difficulties involved in binding tens of thousands of disks into magazines heading for newsstands. It's an exciting undertaking, and we'll be anxiously awaiting the results of the first newsstand sales. Watch for the premiere issue of COMPUTE!'s Atari ST Disk & Magazine in September at your local newsstand that handles COMPUTE! and COMPUTE!'s GAZETTE. We have every hope that it will become a collector's item.

You'll find complete details of our announcement on page 3 of this issue. On page 2 you'll also find a rather interesting contest announcement. We're offering $10,000 in prizes for the very best Atari ST programs and articles. Good luck!

Commodore 64 Forever

At this June's Consumer Electronics Show in Chicago, Commodore plans to unveil something that may seem ho-hum to many people. In an age of 16/32-bit Amigas and STs with megabytes of memory, Commodore is preparing to announce a revamped version of the Commodore 64—basically the same computer in shiny new wrappings. Dubbed the Commodore 64C, it will be a fully compatible 64 in a Commodore 128-style case. Enclosed in the package will be a floppy disk containing a terminal program for accessing the QuantumLink information service, and GEOS, the graphics-oriented operating system and user interface. Expected price: between $160 and $180.

This may not seem too exciting—unless you're a Commodore enthusiast or someone searching for an inexpensive home computer system. From our viewpoint, it's the most exciting 64-related announcement in the past three years. Loud and clear, it broadcasts three important messages:

1. Despite its commitment to establishing the Amiga as its flagship personal computer, Commodore is not abandoning the millions of 64 owners. The Commodore 64C shows that Commodore is determined to continue its support of what has become the world's most popular home computer.

2. The Commodore 64 market will remain a significant source of revenue for software developers, and may even keep expanding.

3. As the bundling of GEOS shows, the 64 is still evolving, growing more powerful and easy to use, and is an exceptional value for people who need a functional computer system for under $500.

Like Apple's slogan when it introduced the Apple IIc—"Apple II Forever"—Commodore is declaring, in effect, "Commodore 64 Forever."

Forever is a long time, and we don't really think the 64 will be around quite *that* long. Still, Commodore's renewed commitment to the 64 reassures those who have wondered if their computers would soon be "orphans." COMPUTE! has received many letters from readers who feared that the 64 market would dry up and vanish now that Commodore is preoccupied with the Amiga and 128. And actually, as we reported several issues ago, Commodore did attempt to shut down 64 production more than once last year. But each time, the unabated hunger for this four-year-old machine swamped Commodore with orders, and the company was forced to restart production and rethink its strategy. The 64 refuses to die.

So Commodore is taking advantage of the situation by bringing the computer up-to-date without sacrificing its compatibility with the thousands of programs and peripherals already on the market. Here is what Commodore plans to announce at CES:

• The 64C in a more professional-looking Commodore 128-style case (minus the 128's numeric keypad);

• A bundled disk containing GEOS (Graphics Environment Operating System) and QuantumLink software. GEOS is patterned after the desktops found on the Macintosh, Atari ST, and Amiga—windows, icons, pull-down menus, bit-mapped graphics, and multiple onscreen type fonts. GEOS includes several integrated application programs and desk accessories, including GEOpaint, GEOwrite, a calculator, notepad, and clock. In addition, GEOS significantly speeds up disk access without modifying the 1541 drive. (For more details on GEOS, see our Winter CES report in the April 1986 issue of COMPUTE!.)

• On the flip side of the disk, 64C buyers will get the special terminal software necessary to access QuantumLink, the online communications service specially tailored to Commodore users.

• The 1541 drive will also get a sleek new case to match the 64C.

• Memory expansion up to 128K and 512K RAM for the 64 and 128.

• A 3½-inch floppy disk drive for the 64 and 64C, priced around $225.

All in all, it's an interesting series of announcements, and an encouraging development for Commodore 64 enthusiasts everywhere.

Robert Lock

Robert C. Lock,
Editor-In-Chief

Tom R. Halfhill

Tom R. Halfhill
Editor

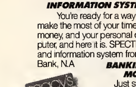
IF YOU'RE READY FOR HOME BANKING BUT YOUR BANK ISN'T...

YOU'RE READY FOR CHASE.

NEW YORK'S PREMIER HOME BANKING AND INFORMATION SYSTEM.

You're ready for a way to make the most of your time, your money, and your personal computer, and here it is. SPECTRUM,℠ the home banking and information system from The Chase Manhattan Bank, N.A.

BANKING JUST GOT A LOT MORE PERSONAL.



Just switch on your PC, and you've got the undivided attention of a banker, a broker, and a bookkeeper. Right in your own home, your office, or wherever. 24 hours a day, 7 days a week. With complete security.

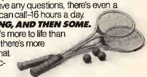
IT'S WHAT HOME BANKING SHOULD BE. EASY.

Touch a button to check your balances; transfer funds between checking, savings, money market and other accounts; pay bills to anybody, anywhere; and keep records. Touch another and you've got key financial planning information to set goals...and meet them.

Touch still another and a complete investment menu is at your service: check the market, get the bottom line on over 4600 companies with S&P Online™ from Standard & Poor's Corporation, track your portfolio, even trade stocks and options at a discount through Rose & Company, a Chase affiliate.

If you do have any questions, there's even a real person you can call—16 hours a day.

HOME BANKING, AND THEN SOME.



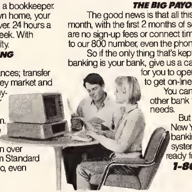
Since there's more to life than banking and finance, there's more to SPECTRUM than that.

Through our electronic bulletin board, you can find a squash partner, a recipe for pasta al pesto, maybe even a ski house share up in Vermont!

THE BIG PAYOFF.

The good news is that all this starts at just $5 a month, with the first 2 months of service *free*. There are no sign-up fees or connect time charges, and, thanks to our 800 number, even the phone calls are free.

So if the only thing that's kept you from home banking is your bank, give us a call. We'll make it easy for you to open an account with us to get on-line with SPECTRUM.



You can still use your other bank for other financial needs.

But if you're ready for New York's premier home banking and information system, chances are, you're ready for Chase.

1-800-522-7766.

CHASE

Publisher	James A. Casella
Founder/Editor in Chief	Robert C. Lock
Senior Editor	Richard Mansfield
Managing Editor	Kathleen Martinek
Executive Editor	Selby Bateman

Editor	Tom R. Halfhill
Assistant Editor	Philip Nelson
Production Director	Tony Roberts
Production Editor	Gail Cowper
Editor, COMPUTE!'s GAZETTE	Lance Elko
Technical Editor	Ottis R. Cowper
Assistant Technical Editor	George Miller
Program Editor	Charles Brannon
Assistant Editor, COMPUTE!'s GAZETTE	Todd Heimarck
Assistant Features Editor	Kathy Yakal
Programming Supervisor	Patrick Parrish
Editorial Programmers	Tim Victor, Kevin Mykytyn
Submissions Reviewer	Mark Tuttle
Programming Assistants	David Florance, David Hensley
Executive Assistant	Debi Nash
Administrative Assistants	Julia Fleming, Iris Brooks, Mary Hunt, Sybil Agee
Associate Editors	Jim Butterfield, Toronto, Canada; Harvey Herman, Greensboro, NC; Fred D'Ignazio, Birmingham, AL; David Thornburg, Los Altos, CA
Contributing Editor	Bill Wilkinson

COMPUTE!'s Book Division	
Editor	Stephen Levy
Assistant Editors	Gregg Keizer, Ann Davies
Director, Book Sales & Marketing	Steve Voyatzis

Production Manager	Irma Swain
Art & Design Director	Janice R. Fary
Assistant Editor, Art & Design	Lee Noel
Mechanical Art Supervisor	De Potter
Artists	Debbie Bray, Dabney Ketrow
Typesetting	Terry Cash, Carole Dunton
Illustrator	Harry Blair

Director of Advertising Sales	Peter Johnsmeyer
Associate Advertising Director	Bernard J. Theobald, Jr.
Production Coordinator	Kathleen Hanlon

Promotion Assistant	Caroline Dark

Customer Service Manager	Diane Longo
Dealer Sales Supervisor	Orchid Tamayo
Individual Order Supervisor	Cassandra Green
Receptionist	Anita Armfield
Warehouse Manager	John Williams

Data Processing Manager	Leon Stokes

James A. Casella, President
Richard J. Marino, Vice President, Advertising Sales
Chris Savine, Director, Finance & Planning

COMPUTE! Publications, Inc. publishes

COMPUTE! The Journal for Progressive Computing

COMPUTE!'s GAZETTE For VIC-20 And Commodore 64 Personal Computers

COMPUTE! Books

COMPUTE!'s GAZETTE DISK

COMPUTE!'s Apple Applications Special

Editorial offices	324 West Wendover Avenue, Suite 200, Greensboro, NC 27408 USA
Corporate offices	825 7th Avenue, New York, NY 10019, 212-265-8360
Customer Service	800-346-6767 (In NY 212-887-8525)
Hours	9:30 A.M.–4:30 P.M. Monday–Friday

Coming In Future Issues

Tightrope: A Game of Taut Reflexes For Commodore 64/128, Atari, Apple, Amiga, And IBM PC/PCjr

Sprite-32 For Commodore 64: How To Display 16 Or 32 Sprites

Softball Statistics For Atari ST

The Logical Alternative For Atari 400/800/XL/XE: Replace Slow IF-THENs With Fast Logic

Apple ProDOS Protector And DOS 3.3 Guardian Angel: Protect Your Programs From Prying Eyes

Commodore 128 Machine Language: A New Series By Jim Butterfield

Foolproof Input For Amiga BASIC

Subscription Orders

COMPUTE!
P.O. Box 10954
Des Moines, IA 50340

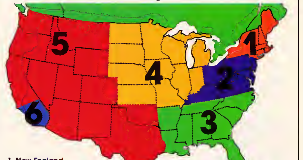
TOLL FREE Subscription Order Line
800-247-5470
In IA 800-532-1272

COMPUTE! Subscription Rates (12 Issue Year):

US	(one yr.)	$24
	(two yrs.)	$45
	(three yrs.)	$65
Canada and Foreign Surface Mail		$30
Foreign Air Delivery		$65

Member ABC Audit Bureau of Circulations   MPA Magazine Publishers Association

Advertising Sales



1. New England
Jonathan Just
Regional Manager
212-315-1665

2. Mid Atlantic
Jonathan Just
Regional Manager
212-315-1665

3. Southeast & Foreign
Harry Blair
919-275-9809

4. Midwest
Gordon Benson
312-362-1821

5. Northwest/Mountain/Texas
Phoebe Thompson
Dani Nunes
408-354-5553

6. Southwest
Ed Winchell
213-378-8361

Director of Advertising Sales:
Peter Johnsmeyer

Associate Advertising Director:
Bernard J. Theobald, Jr.

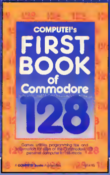
COMPUTE! Home Office 212-887-8460.

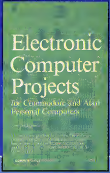
Address all advertising materials to:
Kathleen Hanlon
Advertising Production Coordinator
COMPUTE! Magazine
324 West Wendover Avenue
Suite 200
Greensboro, NC 27408

The COMPUTE! subscriber list is made available to carefully screened organizations with a product or service which may be of interest to our readers. If you prefer not to receive such mailings, please send an exact copy of your subscription label to: COMPUTE! P.O. Box 10955, Des Moines, IA 50950. Include a note indicating your preference to receive only your subscription.

Two Exciting New Books

from COMPUTE!





COMPUTE!'s First Book of the Commodore 128

A spectacular collection of articles and programs exclusively for the Commodore 128 *in 128 mode.*

Edited

The editors at COMPUTE! Publications have collected some of the best games, programs, and tutorials for the Commodore 128 from *COMPUTE!* and *COMPUTE!'s Gazette*, plus some never-before-published articles and programs. Learn how to create windows, program sound, and make disks autoload. You'll even find a map of all the important memory locations. There's something for every 128 user. All programs run in 128 mode. *A disk is available which includes programs in the book, $12.95.*

$14.95 ISBN 0-87455-059-9

Electronic Computer Projects

Learn how to build all kinds of new devices to interface with your computer from inexpensive, available parts.

For the Commodore 64, 128, VIC, and any eight-bit Atari personal computer.

Soori Sivakumaran

This introduction to digital electronics and computer interfacing is the easy way to learn how computers interact with the outside world. Using a Commodore 64, 128, VIC, or any eight-bit Atari computer and *Electronic Computer Projects*, you'll be guided through the steps to building a joystick, light pen, game paddle, and numerous other devices. And since each project is independent from the others, you can choose only those projects that interest you. All the projects can be built at home and most require fewer than half a dozen parts.

$9.95 ISBN 0-87455-052-1

Visit your local book or computer store for these new titles. Or order directly from COMPUTE! Books. Call toll-free 800-346-6767 (in NY 212-887-8525) or write COMPUTE! Books, P.O. Box 5038, F.D.R. Station, New York, NY 10150.

Please include $2.00 per book ($5.00 air mail) for shipping and handling. NC residents add 4.5 percent sales tax. Allow 4–6 weeks from receipt of order for delivery.

COMPUTE! Publications, Inc. abc
Part of ABC Consumer Magazines, Inc.
One of the ABC Publishing Companies
825 7th Avenue, 6th Floor, New York, NY 10019
Publishers of COMPUTE!, COMPUTE!'s Gazette, COMPUTE!'s Gazette Disk, COMPUTE! Books, and COMPUTE!'s Apple Applications

COMPUTE! books are available in the U.K., Europe, the Middle East, and Africa from Holt Saunders, Ltd., 1 St. Anne's Road, Eastbourne, East Sussex BN21 3UN, England, and in Canada from McGraw-Hill, Ryerson Ltd., 330 Progress Ave., Scarborough, Ontario, Canada M1P 2Z5.

Readers Feedback

The Editors and Readers of COMPUTE!

If you have any questions, comments, or suggestions you would like to see addressed in this column, write to "Readers' Feedback," COMPUTE!, P.O. Box 5406, Greensboro, NC 27403. Due to the volume of mail we receive, we regret that we cannot provide personal answers to technical questions.

The Ideal BASIC Style

Some time ago I read a letter in your magazine regarding crunching of program listings and the effect this has on readability. You replied that this was to save memory and magazine space. I would like to suggest a reasonable compromise between readability and the elimination of spaces. In my view, any statement that juxtaposes two letters (for instance, FORT=1TO10 or IFS=5) would benefit greatly from extra spaces (FOR T=1 TO 10 or IF S=5). But if a number follows a letter (as in GOTO600 or THEN470), the statement is understandable even without an extra space. I think DATA should always have a following space so the first value stands out clearly. You are often inconsistent in this, even within the same program listing. As to multiple statements in one line, this in itself creates no problems and is necessary in some cases. But I don't believe that completely unrelated statements should be put on the same line simply to fill up the line.

Line numbering is another area. You use the time-honored decade numbering (line increments of ten), which is fine when developing a program. Finished programs are usually renumbered for neatness, but I don't see why nine skipped numbers are necessary. I suggest that you use every other number instead (1, 3, 5, and so on). This would allow someone to insert a STOP or GOTO while checking for typing errors or making minor alterations. The big advantage of this system is that decade numbers could have special meanings as important entry points or the beginning of a new group of closely related statements. For example, a complex FOR-NEXT loop might use several lines, then jump to the next decade line number for a new group of related statements. It would be much easier to follow and understand the flow pattern.

I also feel there could be at least partial standardization of some of the most common variable names. For instance, the variables I, J, and K are ordinarily used as "junk" variables (counters within loops, and so on). The variables X and Y are frequently used to specify horizontal and vertical coordinates. But many others are commonly used as well: SA for *starting address*, EA for *ending address*, CK for *checksum*, and so forth. You could publish a list of suggested variable names and encourage programmers to stick to it.

Don R. King

As long as programmers use BASIC, there will be discussions about what sort of style and structure BASIC programs ought to have. The reason for the controversy is familiar. BASIC imposes few structural constraints on the programmer, so the language is easy to learn and works well for improvisational programming and quick experiments. But its lack of structure also makes it possible to write tangled, illogical "spaghetti" code. Since BASIC doesn't force you into a predetermined mold, a program can take nearly any form. More structured programming languages such as Pascal generate more readable code, but demand more forethought on the programmer's part.

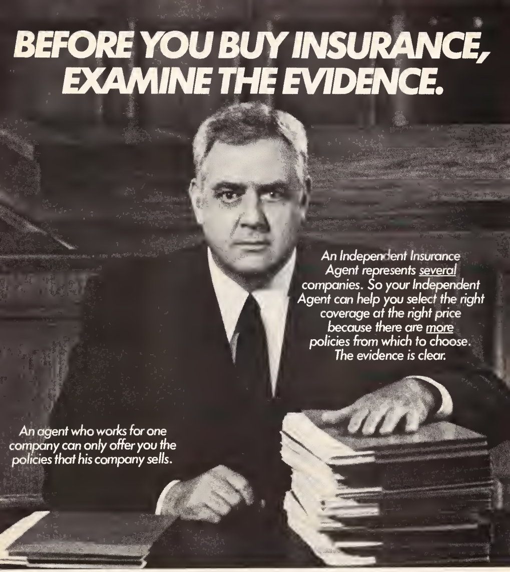
Most of the programs we publish are submissions from readers.Generally, we modify these programs only to eliminate any bugs that appear during testing or to add functional improvements. Any time you change someone else's program, you increase the likelihood of inadvertently creating new bugs which even the author may not have anticipated. Given the number of programs we publish and the constraints of monthly deadlines, it's not practical for us to rewrite working programs merely to improve their readability.

A carefully planned numbering scheme can add to a program's readability. But our programs are meant to be typed in from a printed listing as well as studied. So we need to do everything possible to help readers type the programs without errors. Numbering in regular increments makes it easier to keep your place in the program than if the increments changed at unpredictable intervals. Uniform numbering also helps readers spot lines that have been left out altogether (a typing error that no proofreader program can catch). However, sometimes even the simple act of renumbering a program can introduce new bugs—as has happened to us in the past.

It's also true that if everyone followed the same stylistic conventions, BASIC programs would be more readable. The difficult part is getting programmers to go along with the conventions you choose, especially considering that each version of BASIC has its own peculiarities. For instance, Commodore BASIC doesn't require spaces after keywords (and omitting spaces speeds program execution), but some other versions of Microsoft BASIC insist on a separating space. Other BASICs, such as Atari BASIC, automatically insert spaces for readability if you leave them out.

Different dialects of BASIC also include different keywords. For instance, NAME is a legal variable name in Commodore BASIC, but it's treated as a reserved word in IBM BASICA and Amiga BASIC. In Commodore and Apple BASICs, only the first two characters of the variable name are significant, and you may not embed keywords in variable names. But IBM, Atari, and Amiga BASICs permit long, descriptive variable names such as MousePosition *or* MenuFlag *which can include embedded keywords. The list of differences goes on and on. Given the diversity among BASIC dialects and the absence of standardization, any list of preferred variable names would have to be exceedingly general and geared toward the lowest common denominator.*

As time goes by, Microsoft BASIC seems to be taking over as the de facto standard for the language. Newer, more powerful computers such as the Macintosh, Atari ST, and Amiga all offer versions of BASIC that more closely resemble IBM BASICA. With the exception of graphics and sound statements, which are necessarily hardware-specific, a program that runs on the IBM, Mac, or Amiga will probably run on any of the others with only slight modifications. If this trend continues, we may someday reach the point where BASIC style becomes more homogeneous.



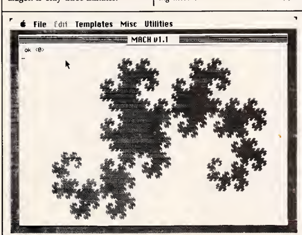
THE MORE-THAN-ONE-COMPANY INSURANCE AGENT.

You'll find the Independent Insurance Agent nearest you in the Yellow Pages

YOUR Independent Insurance AGENT SERVES YOU FIRST ®

Faster Fractals In Forth

I enjoyed reading Paul Carlson's article on fractal graphics for the IBM PC/PCjr (COMPUTE!, March 1986). His explanations were very clear. But it must have been a real trial for him to develop the BASIC version of the "Eight Thousand Dragons" program. We would like to show the beauty of fractals when written with a language that supports recursion. Here is an example of Forth code that does the same thing. It's written for *Mach1*, our Forth compiler for the Apple Macintosh and Atari ST. The execution time for a fourteenth-degree dragon is only three minutes.




```
: Drag RECURSIVE { x1 y1 x2 y2 x3 y3 k l x4 y4 x5 y5 }
      PAUSE
      k 0=                    (if k=0 just draw and return)
      IF                      (else continue breaking lines down)
            k 1- -> x4
            x1 x2 + 2/ y2 y1 - 2/ - -> x4
            y1 y2 + 2/ x2 x1 - 2/ + -> y4
            x2 x3 + 2/ y3 y2 - 2/ + -> x5
            y2 y3 + 2/ x3 x2 - 2/ - -> y5
            x1 y1 x4 y4 x2 y2 k Drag
            x2 y2 x5 y5 x3 y3 k Drag
      THEN ;

:Dragon { iters --}          ('14 dragon' gives the best results)
      CLS
      100 190 CALL MoveTo                ( place pen at beginning)
      100 190 228 62 356 190 iters Drag  ( start with initial seed)
```

Terry Noyes

Although recursive routines (program segments that call themselves) are ordinarily taboo in BASIC, they're not only feasible, but encouraged in other languages such as Logo and Forth. Besides speeding execution, recursion produces compact, elegant code, as this example shows. Thanks for the demonstration.

Reading The Atari Touch Tablet In BASIC

I am currently working on an Atari program that lets me create high-resolution drawings in graphics mode 15. However, the drawing should be done with the Atari Touch Tablet. How can a program read the Touch Tablet coordinates?

Peter Hinz

Reading coordinates from the Atari Touch Tablet is very easy in Atari BASIC. The Touch Tablet returns the same values as paddle controllers, and Atari BASIC contains a function called PADDLE for reading these controllers. Use PADDLE(0) to read the horizontal position of the stylus on the tablet, and PADDLE(1) to read the vertical position (assuming that the tablet is plugged into controller port 1). Both functions return values ranging from 1 to 228. When nothing is touching the tablet surface, these functions return the value 228.

Reading the Touch Tablet buttons is just as easy. Use the PTRIG(0) function to read the left button, and PTRIG(1) to read the right button (again, assuming that the tablet is plugged into port 1). When a button is pressed, these functions return a value of 0. Otherwise, they return a value of 1.

The button on the Touch Tablet's stylus works a little differently. To detect this button press, use the STICK(0) function (normally intended for reading a joystick). If the stylus button is pressed, STICK(0) returns a value of 14. Otherwise, it returns the value 15.

The following example program prints the tablet coordinates on the screen along with messages when any of the buttons are pressed:

```
BN 10 X=PADDLE(0):Y=PADDLE(1
      )
MH 20 PRINT X,Y
ID 30 IF PTRIG(0)=0 THEN PRI
      NT "LEFT BUTTON PRESSE
      D"
NI 40 IF PTRIG(1)=0 THEN PRI
      NT "RIGHT BUTTON PRESS
      ED"
HL 50 IF STICK(0)=14 THEN PR
      INT "STYLUS BUTTON PRE
      SSED"
AA 60 GOTO 10
```

Safe Zones In IBM BASIC

Is there any way to store a few characters or flags in the IBM PC's memory that will survive the BASIC RUN command? I want my program to be able to "learn" as it runs and remember what it has learned each time it is run.

H. Beck

IBM BASIC's CLEAR command gives you the ability to create a safe area of RAM of almost any size. Besides deleting all variables, CLEAR controls the amount of memory available to BASIC. By adding a comma and a parameter to the CLEAR command, you can make the BASIC workspace smaller than usual, reserving the extra memory for yourself. The workspace is initially 65,536 bytes, but it's easy to reserve some memory at the top of that space. Use this format:

CLEAR ***,workspace***

where workspace *is a number less than 65536. To calculate the correct value, subtract from 65536 the number of bytes you want to protect. For instance, the command CLEAR ,65280 reserves the last 256 bytes (65536 − 256 = 65280) of BASIC workspace for your use.*

When you type RUN after a CLEAR statement like this, the size of the workspace is reset to its default but the data in the reserved area is not affected. As long as the next program begins with a similar CLEAR statement, it can PEEK into the reserved area and find the values that the previous program POKEd there. Here's a

simple program that stores some values in a 256-byte reserved area:

```
10 CLEAR,65280
20 FOR A=0 TO 255
30 POKE A+65280,A
40 NEXT
```

After you run the program, enter and run this program to read the stored values back.

```
10 CLEAR,65280
20 FOR A=0 TO 255
30 PRINT PEEK (A+65280)
40 NEXT
```

Scanning The 128's ALT Key

Please tell me how to read the ALT key on the Commodore 128.

J. C. Vollmer

The 128's ALT key cannot be polled like the other keys. Instead, your program must PEEK location 211, where the system stores information about five special keys: SHIFT, CONTROL, ALT, CAPS LOCK, and the Commodore key. When you press one of these keys, it sets a certain bit in this location:

Key	Bit	PEEK(211) Value
SHIFT	0	1
Commodore	1	2
CONTROL	2	4
ALT	3	8
CAPS LOCK	4	16

Thus, if PEEK(211) equals 8, you know the ALT key is pressed. Since each key has its own signal bit, the values from location 211 are additive. If PEEK(211) equals 9, for instance, both SHIFT and ALT are pressed. When SHIFT and CONTROL are pressed, location 211 holds 5, and so forth.

Correction For Casio Review

I enjoyed reading your January 1986 review of *The Music Shop for MIDI* and the Casio CZ-101 synthesizer. In fact, I became inspired and bought the same system, after having exhausted the 64's musical capabilities. You mentioned a problem with accessing all the features of the CZ-101 synthesizer. Perhaps I obtained an updated version of *The Music Shop for MIDI* because I have not had the same experience. All 48 timbres can be accessed (the basic presets plus those in internal memory or cartridge memory). In addition, four-voice polyphonic music is possible within the program using the programmable MIDI features (channeling the four solo voices). Casio's COSMO series of synthesizers, which includes the CZ-101, is capable of playing up to eight timbres (four on the CZ-101) on a single slave unit. Have you looked into other types of MIDI software presently available?

Please continue your articles on computer music.

Eric Habeck

Thanks to you and to Don Williams (the programmer who created The Music Shop *and* The Music Shop for MIDI*) for alerting us to the misstatement. The 64 is very popular with musicians and sound enthusiasts because of its low cost and built-in sound capabilities. As the MIDI standard becomes more widely accepted, we're likely to see even more in the way of music software for the 64. We'll continue to review new products as time and space permit.*

Customizing *SpeedScript 3.0*

The "*SpeedScript* Customizer" program that appeared in the September 1984 issue of COMPUTE!'s GAZETTE allows you to change the default settings and formatting commands in *SpeedScript 1.0* to fit your own preferences. But that program doesn't work with *SpeedScript 3.0* or 3.2. Do you have an update of the program or the necessary POKEs to allow the same customization for the most recent versions of *SpeedScript*?

Bruce Patten

It just so happens that another reader wrote in with the very information you're seeking:

"*SpeedScript* Customizer" doesn't work with *SpeedScript 3.0* or 3.2. But I have discovered the POKEs for customizing all of the same parameters for the newest versions of *SpeedScript*. Here are the default (normal) values and the locations that control them:

Default Location	Value	Parameter
5722	5	Left margin
5723	75	Right margin
5724	66	Page length
5725	5	Top margin
5726	58	Bottom margin
5727	1	Spacing
5728	1	Wait (1=go ahead)
5729	1	@ start numbering pages at (LSB)
5730	0	(MSB)
5731	1	? starting printing at page (LSB)
5732	0	(MSB)
5733	80	x columns across
5734	27	1 printkey 1
5735	14	2 printkey 2
5736	15	3 printkey 3
5737	18	4 printkey 4

To customize your program, load *SpeedScript 3.0* or 3.2 into memory, then POKE the desired values into the appropriate memory locations. For example, POKE 5722,3 makes the left margin setting default to 3 instead of 5. Then save the program using a different name. For instance, I have a frequent need to print postcards, so I set the left margin at 3, right margin at 35, columns at 40, top margin at 3, bottom margin at 18, and page length at 2. If you want to start numbering pages or start printing pages at a page lower than 256, POKE the desired value in the first of the two locations indicated. For instance, to start numbering pages at page 3, you would POKE 5729,3. To start at a page higher than 255, you must POKE two values in low byte/high byte format. The low byte of the value goes into the lower location.

Allen Perkins

Thank you for the information. As mentioned in the original SpeedScript *article, most of these settings have to do with formatting hardcopy printouts.*

Apple RESET Vectoring

Is there any way to make the Apple II jump to a specific machine language subroutine after the RESET key has been hit?

Jose A. Colon Olivo

This can done by changing the two-byte RESET vector at location 1010 ($03F2). The most direct way to alter the vector is to POKE the starting address of your machine language program into locations 1010–1011 in low byte/high byte format, then update these pointers with CALL −1169. When you hit RESET, the Apple checks the vector, goes to the indicated location, and runs your program. As an example, suppose you wish to execute the following routine which prints an A on the screen upon RESET:

```
0300 LDA #$C1
0302 JSR $FDF0
0305 JMP $03D0
```

The first step is to determine the high and low bytes of the starting address in decimal. The hexadecimal number $0300 is expressed as decimal 768. So, the high and low bytes of the starting address are:

```
HI=INT(768/256)=3
LO=768−HI*256=0
```

Next, POKE the address values in 1010 and 1011, and execute the CALL to update the pointers:

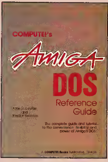
```
POKE 1010,LO
POKE 1011,HI
CALL −1169
```

When you hit RESET, the routine executes. Notice that the ML routine ends with JMP $03D0. Because the routine is jumped to directly, it leaves no return address on the microprocessor's stack. If it had ended with an RTS, you'd wind up back in the machine language monitor after it's done. To avoid this unwanted result, you must exit with a JMP to the BASIC soft reentry point at $03D0.

AMIGA

SUPPORT FROM COMPUTE! BOOKS

Everything for the Amiga. From BASIC beginner's guides to advanced programming handbooks, COMPUTE! offers you information-packed tutorials, reference guides, programming examples, ready-to-enter applications, and games to help you develop your computing skills on Commodore's Amiga.



COMPUTE!'s AmigaDOS Reference Guide

Arlan R Levitan and Sheldon Leemon

A comprehensive tutorial and reference guide to the powerful AmigaDOS—the operating system underlying the Workbench and Intuition—this book offers information useful to every Amiga owner It defines and illustrates all DOS commands, and shows you how to create file directories, access peripherals, run batch file programs, and avoid "disk shuffle." The screen- and line-oriented text editors are explained in detail Numerous examples and techniques explain how to use AmigaDOS to make operating your Amiga both convenient and efficient

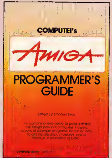
$14.95 ISBN 0-87455-047-5

Elementary Amiga BASIC

C Regena

Here's your introduction to the new and powerful BASIC on the Amiga personal computer The Amiga's impressive graphics, animation, and sound can be unlocked with the right commands, and BASIC is the place to start Complete descriptions of Amiga BASIC's commands, syntax, and organization take you from the beginner level to a full-fledged programmer Plus, the book offers you ready-to-type-in programs and subroutines while showing you how to write your own programs *There is a disk available which includes the programs in the book, $12 95 This title is also available as a book/disk combination for $29 95 (057-2)*

$14.95 ISBN 0-87455-041-6



COMPUTE!'s Amiga Programmer's Guide

Edited

Your tutorial and reference manual to AmigaDOS, BASIC, Intuition, and other important software tools which accompany the new Amiga, *COMPUTE!'s Amiga Programmer's Guide* is a clear and thorough guide to the inner workings of this fascinating new-generation computer The great speed of its 68000 microprocessor, coupled with the versatility of the Amiga-specific graphics and sound, makes the Amiga one of the most powerful computers available today. This book is the key to accessing the Amiga's speed and power

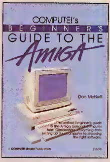
$16.95 ISBN 0-87455-028-9

Advanced Amiga BASIC

Tom R Halfhill and Charles Brannon

This guide to applications programming on Commodore's new Amiga contains everything an intermediate programmer requires to begin creating sophisticated software on this powerful machine, including several ready-to-type-in programs. Clear, yet comprehensive documentation and examples cover advanced BASIC commands, designing graphic applications, generating sound and music, using the Amiga's built-in speech synthesizer, creating a user interface, and programming the computer's peripherals *There is a disk available which includes the programs in the book*, $15.95 (June release)

$16.95 ISBN 0-87455-045-9



COMPUTE!'s Beginners Guide to the Amiga

Dan McNeill

Written in a lively and entertaining style, this book teaches you everything a beginner needs to know to get started quickly with the Amiga from Commodore You will learn about setting up the system, all the most popular types of software, and details about the hardware

$16.95 ISBN 0-87455-025-4

Inside Amiga Graphics

Sheldon Leemon

The Amiga, Commodore's powerful new computer, is an extraordinarily impressive graphics machine Easy to use, the Amiga can produce color graphics and excellent animation You'll find thorough descriptions of the computer's abilities and the hardware required to create a complete graphics system Software, too, is central to the Amiga's power, and complete tutorials show you how to get the most from the machine (June release)

$16.95 ISBN 0-87455-040-8

COMPUTE!'s Kids and the Amiga

Edward H Carlson

The latest in this bestselling series written by Edward Carlson, *COMPUTE!'s Kids and the Amiga*, will acquaint you with BASIC Over 30 sections—all with instructor notes, lessons, assignments, and lively illustrations—entertain and amuse you as you learn to program your new computer Clear writing and concise examples make it easy for anyone—children and adults alike—to painlessly learn BASIC (May release)

$14.95 ISBN 0-87455-048-3

Look for these books at your local book or computer store.
Or order directly from COMPUTE!.
Call toll-free 1-800-346-6767 (in NY 212-887-8525).

Please allow 4–6 weeks for delivery after your order is received.

COMPUTE! Publications, Inc. abc
Part of ABC Consumer Magazines Inc
One of the ABC Publishing Companies
825 7th Avenue 6th Floor New York NY 10019
Publishers of COMPUTE! COMPUTE!'s Gazette COMPUTE!'s Gazette Disk COMPUTE! Books and COMPUTE!'s Apple Applications

COMPUTE! books are available in the U.K., Europe, the Middle East, and Africa from Holt Saunders, Ltd., 1 St Anne's Road, Eastbourne, East Sussex BN21 3UN, England, and in Canada from McGraw-Hill, Ryerson Ltd., 330 Progress Ave., Scarborough, Ontario, Canada M1P 2Z5

The same technique can be used to make the computer run a BASIC program when RESET is hit. Simply POKE the location of the Applesoft RUN routine into the RESET vector and call the ML from the first line of a BASIC program (for instance, with CALL −768). This could be done with the following program line:

10 POKE 1010,102: POKE 1011,213: CALL −1169

In this case, you'd want to return to BASIC by ending the ML routine with RTS rather than JMP $03D0.

Machine Language Division

I own a Commodore 64 and am teaching myself machine language. There is only one problem preventing me from completing my first useful program—I can't write a program to divide by ten. The objective is to take a number in the range 0–255 from memory and break it into each of its decimal parts. For example, 255 would break down into the digits 2, 5, and 5.

Kevin Owens

The 6502/6510 instruction set does not include a division instruction of any sort. Although it's possible to construct an ML routine for division, it is much easier and faster to use a method called successive subtraction. *Here is the basic idea: Each power of ten from highest to lowest (in this case from 100 to 10 to 1) is subtracted from the number until you determine that the number has become negative. Each digit of the number is derived by counting the number of subtractions.*

Below is the source code for a short program that displays any three-digit number at the upper-left corner of the screen. You'll need a machine language assembler to create the object code. (This program is written in PAL assembler format for the Commodore 64. Slight modifications are needed to assemble the code with a different assembler or to make the program work on a different 6502 computer.) Once the program is assembled, enter SYS 828: The program displays all the numbers from 0 to 255 in succession. The delay loop in line 130 gives you time to read each number. To see how fast numbers can be converted and displayed, remove this line and reassemble the program.

```
10 SYS 700:.OPT OO:*=828
20 TEMP = 2
30 LDA #0:STA 53281:LDA #147:J
   SR $FFD2
40 LDA #1:STA 53281
50 LDA #0:STA TEMP
60 START INC TEMP:LDA TEMP
70 LDX #0
80 SUBAGAIN LDY #255
90 SUBMORE INY:SEC:SBC DIGITS,
   X
100 BCS SUBMORE
110 ADC DIGITS,X
120 PHA:TYA:ORA #48:STA 1024,X
    :PLA
130 INX:CPX #3:BNE SUBAGAIN
140 LDY #8:LDX #0:WT DEX:BNE W
    T:DEY:BNE WT
150 JMP START
160 DIGITS .BYT 100,10,1
```

Atari BASIC Bugs

I have a very serious problem with my 16K Atari 600XL. Sometimes right after I've entered a line, the computer locks up and the only key that will work is SYSTEM RESET. But after I press RESET and enter another line, the computer locks up again.

Tak Lee

You're experiencing the latest incarnation of the infamous Atari BASIC lockup bug. This bug afflicts two versions of Atari BASIC: the original version, known as revision A, which was supplied as a cartridge for the 400, 800, and 1200XL; and revision B, which is built into the 600XL and 800XL computers. The lockup bug takes a slightly different form in these two versions of BASIC. In revision A, BASIC is unable to delete (move downward in memory) a block of memory whose size is an exact multiple of 256 bytes. Most users encounter the bug in the form of a keyboard lockup after deleting program lines, but it can affect the movement of strings as well. To illustrate, type in the following program:

```
AO 10 DIM A$(256),B$(256)
FD 20 FOR A=1 TO 256:A$(A,A)
      ="B":NEXT A
GL 30 B$=A$
ID 40 PRINT A$:PRINT B$
```

This creates a string variable, A$, that consists of 256 B characters. Then it makes B$ equal to A$. When the program runs, you would expect it to print the letter B 512 times. Type RUN to see what happens instead. If you have revision A BASIC, the first 256 characters are correct, but the remaining characters are garbage. This occurs because BASIC's memory move routine was unable to move the value of A$ into B$ correctly. To confirm that the bug applies only to blocks of memory in multiples of 256, try changing the number 256 in lines 10 and 20 to some other value.

This bug was corrected when revision B BASIC was prepared for the 600XL and 800XL. However, Atari's programmers got carried away and applied the same correction to a routine which didn't need fixing—the routine which inserts (moves upward) blocks of memory, which happens when you add a BASIC program line. As a result, revision B BASIC has its own lockup bug which rears its ugly head when program lines are inserted instead of deleted. Ironically, the revision B bug may occur even more often than the old one—you add program lines more often than you delete them. As an example, turn your 600XL or 800XL off and back on, then enter the following line in direct mode:

DIM A$(249):A$="TRASH"

Enter PRINT A$ to see the variable value. Now enter this program line:

10 PRINT "THIS IS A TEST"

Before doing anything else, try to print the string again:

PRINT A$

If you have revision B, your computer should be locked up, and pressing RESET won't recover. For more details, see Bill Wilkinson's "Insight: Atari" columns in the May and June 1985 issues of COMPUTE!.

The line-editing bug is not the only problem in early versions of Atari BASIC. Here's a list of some of the other bugs in revision A:

1. Because the value −0 is interpreted incorrectly, printing −0 yields garbage.

2. There's a problem with the precedence of the NOT operator which causes it to give unpredictable results in some cases. Type the following statement in direct mode when you have no other program in memory: PRINT NOT NOT 1.

3. LOCATE and GET statements may corrupt the internal buffer pointer. This can cause difficulties when trying to READ from DATA statements or when using the VAL function. If you have trouble with READ or VAL, use the statement X$=STR$(0) to reset the buffer pointer after a GET or LOCATE.

Revision B BASIC corrects some of the bugs from revision A, but also adds a few of its own. Here are some revision B bugs.

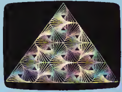
1. BASIC adds 16 bytes to the end of a program every time you LOAD or CLOAD it. After loading and saving the same program several times, you'll find that it has grown substantially. To remove the extra bytes, LIST the program to disk or tape and ENTER it back into memory. To avoid the problem, always use LIST/ENTER instead of SAVE/LOAD.

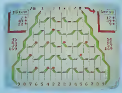
2. CLOAD and CSAVE commands fail to turn off the sound after they're done. Use END or the statement SOUND 0,0,0,0 to silence it.

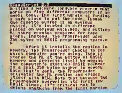
3. Occasionally, an ERROR 9 (array or string DIM error) wrongly occurs in a program line that contains a DIM statement. You may be able to fix this condition by LISTing the program to disk and ENTERing it again.

One solution is to get the newest Atari BASIC cartridge, revision C, from Atari Corp., Customer Product Service, P.O. Box 61657, Sunnyvale, CA 94088. It costs $15. This is the same BASIC built into the Atari 65XE and 130XE models. ©

GET UP TO 200 FUN-FILLED PROGRAMS EACH YEAR–
when you subscribe now to COMPUTE!







Subscribe to COMPUTE! today through this special introductory money-saving offer, and you'll be getting a lot more than just another computer magazine. That's because each issue of COMPUTE! comes complete with up to 20 all-new, action-packed programs.

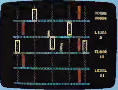
Subscribe now and you can depend on a steady supply of high quality, fun-filled programs like Hickory Dickory Dock, Switchbox, TurboDisk, Home Financial Calculator, Turbo Tape, SpeedScript, SpeedCalc, and hundreds of other educational, home finance, and game programs the entire family can use all year long.

The superb programs you'll find in each issue are worth much, much more than the low subscription price.

And there's more to COMPUTE! than just exciting new programs. Month after month, COMPUTE!'s superb articles deliver the latest inside word on everything from languages to interfaces...from programming to disk drives.

Whether you're a novice or an experienced user, COMPUTE! is the magazine for you. So subscribe today. Return the enclosed card or call 1-800-247-5470 (in Iowa, 1-800-532-1272).

Do it now.



ACT NOW AND SAVE!

Getting Down to BASICs

In one form or another, the BASIC programming language has been around since before the dawn of personal computing. Easy to learn, simple to use, and useful for a wide variety of tasks, BASIC has opened the doors of computer programming for millions of people. Now, as a new generation of personal computers emerges, BASIC is continuing to evolve as a friendly and functional computer language—with a new look and feel.

Why BASIC? Why has this 22-year-old programming language—Beginner's All-purpose Symbolic Instruction Code—grown so immensely popular as an introduction to computer programming and as a general language? Despite its inherent limitations and numerous critics, BASIC remains the most widely taught and used language among computerists today.

The answers to these questions go back to a period before the advent of personal computing, in fact, years before a microcomputer was ever built. In the 1960s, the large mainframe computers ran programs by processing batches of punch cards—a system known as *batch processing*. These mainframes were machines that required a corps of trained operators to serve relatively untutored users. That average people would someday own and program powerful computers which fit on a desktop was unimaginable.

In those days, you didn't interact with a computer so much as present punch card offerings to it, and then wait for the computer to tell the computer operator to tell you the answer. More often than not, the first result was that a mistake somewhere in your batch of cards made an answer impossible until you corrected the error and then resubmitted the batch for another run. Not only was the process tedious, but it also meant waiting in line for access to the computer. A computer could serve only one user at a time.

Today, we tend to take for granted our ability to communicate quickly and easily with computers. The proliferation of personal computers has meant that individuals can have control over when and how often they work with a computer. And that kind of accessibility means that many people need a relatively easy way to communicate with computers without having to rely on other people as translators.

The original BASIC grew out of a project started back in 1964 by Dartmouth College mathematics professors Dr. John G. Kemeny and Dr. Thomas E. Kurtz. Kemeny and Kurtz were working with students on a timesharing project that would allow several people to gain simultaneous access to the university's mainframe computers. As a part of

the project, Kemeny and Kurtz developed BASIC, from which all subsequent versions have evolved. The professors wanted BASIC to be an easy-to-use general-purpose programming language for their mainframe system that would allow more people to communicate with the computer on their own.

By the 1970s, when the first microcomputers were becoming available, BASIC had come to be regarded as an excellent language for personal programming and had already undergone several mutations. It became one of the first languages implemented on a personal computer when two college students—Paul Allen and Bill Gates—adapted it for the kit-built 4K RAM Altair in 1974. Allen and Gates later went on to found their own software company, Microsoft, Inc. Today, vastly improved descendants of the original Microsoft BASIC are available for almost all personal computers.

Developers of the original BASIC programming language, Dr. Thomas E. Kurtz (left) and Dr. John G. Kemeny, who now market a new version called True BASIC.

As BASIC became more widespread, it evolved in many different directions. Although the most popular version is Microsoft BASIC, none of the dozens of dialects adheres to a single standard. An involved program written in BASIC for one type of computer will rarely run on another type without at least some adjustments. Each version of BASIC embodies the strengths and weaknesses of the computer on which it runs, as well as the additions and deletions of those who adapted it from earlier versions. Some BASICs are so different from each other that they're almost like completely different languages.

Recently, this diversity led Kemeny and Kurtz to introduce what they call *True BASIC* (Addison-Wesley Publishing Company, Reading, Massachusetts) in late 1984. *True BASIC* is available for the IBM PC and compatibles, the Commodore Amiga, and the Apple Macintosh. In part, it's an attempt to deflect some of the criticism which has been aimed at BASIC over the years. Critics of BASIC often decry its lack of *structure*—it's not only possible, but quite easy, to write a BASIC program so disorganized that even the programmer cannot easily decipher it. On the other hand, BASIC's freedom from excessive structure-promoting rules is the very feature which attracts many programmers who prefer a more freeform style. Structured languages tend to encourage the production of more readable code, but also tend to impose more rules on the programmer. The debate over how rigidly structured a programming language should be is unlikely to end anytime soon.

True BASIC definitely leans toward the structured side. In fact, some of its new commands are almost identical to commands in Pascal, a popular structured language. Kemeny and Kurtz hope that *True BASIC*'s structure, speed, error-handling, mouse support, graphics, and easy transportability to other computers will establish it as a new standard.

As personal computers gained popularity, BASIC proved to be a fairly easy language to learn for most people. Since most versions of BASIC are *interpreters*, a programmer can enter a line of BASIC statements and test it immediately. Feedback is rapid because the computer interprets and carries out the commands instantly. But that also means that the computer has to

Color Monitor Sale

Sale

16 COLORS

(Premium Quality)

- Built in Speoker & Audio
- For Video Recorders
- For Smoll Business Computers
- Apple - Commodore -Atori - Aplus 3000 -etc.
- One Yeor Worronty'

(Premium Quality)

- Beoutlful Color Controst
- Hlgh Resolutlon
- Shorp Cleor Text
- Anti-Glore Screen
- 40 Columns x 24 Llnes
- Front Ponel Controls

List $329.00

Sale $139.95*

Add $14.50 Shipping

80 Columns

RGB

Super Hlgh Resolution

13" Color Computer Monitor'

*C64/Atori composite coble $9.95

* C128 RGB/Composite 80 column coble $19.95.

14" RGB & Composite Color Monitor

Allows use of C-128 ond C64 mode - composite ond 80 column RGB mode. Must be used to get 80 columns in color with 80 column computers. Speciolly designed for use with the C128's speciol composite video output, plus green screen only option switch. (odd $14.50 shipping)

List $399.00

Sale $259.95*

14" MAGNAVOX Higher Resolutlon RGB & Composite Monitor (Add $14.50 Shipping)

Sole $279.95*

12" 80 Column Green/Amber Monitor

Super high resolution composite green or omber screen monitor. 80 columns x 24 lines, eosy to reod. Fontostic volue. Limited Quontities.

List $129.00

Sale $79.95*

9" Samsung Hi Res Green Screen Monitor

Super High Resolution 80 column monitor perfect for Apple & Aplus 3000 computers. Fontostic Volue. Very Limited Quontities.

List $129.95

Sale $59.95*

Turn Your Monitor into a TV Set Without Moving Your Computer

Elegont TV Tuner with duol UHF/VHF selector switches goes between your computer ond monitor. Includes mute, outomotic fine tuning ond computer-TV selector switches. Inputs included for 300 ohm, 75 ohm, ond UHF. Con be used with coble TV ond VCR's. Fontostic Volue. Limited Quontities. (Includes loop ontenno for UHF & RCA connecting cobles)

List $129.95

Sale $49.95

15 Day Free Trial - 90 Day Immediate Replacement Warranty

• LOWEST PRICES • BEST SERVICE IN U.S.A. • ONE DAY EXPRESS MAIL • OVER 500 PROGRAMS • FREE CATALOGS

Add $10.00 far shipping, handling and insurance. Illinais residents please add 6¼% tax. Add $20.00 far CANADA, PUERTO RICO, HAWAII, ALASKA, APO-FPO orders. Canadian arders must be in U.S. dollars. WE DO NOT EXPORT TO OTHER COUNTRIES, EXCEPT CANADA. Enclase Cashier Check, Maney Order ar Persanal Check. Allow 14 days for delivery, 2 ta 7 days far phane arders, 1 day express mail! Prices & Availability subject ta change withaut natice.

VISA — MASTER CARD — C.O.D. C.O.D. an phane arders only

PROTECTO

We Love Our Customers

22292 N. Pepper Rd., Barrington, Illinois 60010

312/382-5244 to order

Commodore Software Sale

ORDER TODAY!

Phone Orders — **CALL** 8 to 8 C.S.T. - M-F

312-382-5244

(T) Tape, (C) Cartridge, (D) Disk.

GAMES

Access

☐ 3500 MACH V (C)	$34.95	$20.95
☐ 2128 MACH 128 (D)	49.95	29.95
☐ 0451 BEACH HEAD (D)	39.95	21.95
☐ 3038 BEACH HEAD II (D)	49.95	23.95
☐ 0752 RAID OVER MOSCOW (D)	39.95	26.95
☐ 0118 LEADER BOARD (D)	39.95	24.95

Accolade

☐ 5950 HARDBALL (D)	$29.95	$18.95
☐ 5952 LAW OF THE WEST (D)	29.95	18.95
☐ 5954 FIGHT NIGHT (D)	29.95	18.95
☐ 5956 PSI 5 TRADING CO. (D)	29.95	18.95
☐ 5958 THE DAM BUSTERS (D)	29.95	18.95

Activision

☐ 0761 PITFALL II — LOST CAVERNS (D)	$39.95	$20.95
☐ 0900 SPACE SHUTTLE (D)	37.95	18.95
☐ 0932 ON FIELD FOOTBALL (D)	39.95	20.95
☐ 0936 ON COURT TENNIS (D)	39.95	20.95
☐ 0940 GHOSTBUSTERS (D)	39.95	23.95
☐ 3580 GREAT AMERICAN RD RACE (D)	29.95	18.95
☐ 3582 MASTER OF THE LAMPS (D)	29.95	20.95
☐ 3584 COUNTDOWN/SHUTDOWN (D)	29.95	20.95
☐ 3588 MINDSHADOW (D)	29.95	18.95
☐ 3590 STAR LEAGUE BASEBALL (D)	29.95	20.95
☐ 3592 ALCAZAR (D)	29.95	20.95
☐ 5196 LITTLE PEOPLE PROJECT (D)	34.95	22.95
☐ 5198 FAST TRACKS (D)	34.95	20.95
☐ 5202 GAMEMAKER (D)	39.95	24.95
☐ 3585 COMPLETE FIREWORKS KIT (D)	34.95	22.95
☐ 3612 ALTER EGO (D)	49.95	29.95
☐ 3614 BORROWED TIME (D)	29.95	18.95
☐ 5200 HACKER (D)	29.95	18.95
☐ 1572 STAR RANK BOXING (D)	29.95	20.95

Avalon Hill

☐ 0396 SUPER BOWL SUNDAY (D)	$35.00	$22.95
☐ 3572 SPITFIRE 40 (D)	35.00	22.95
☐ 5138 STATIS PRO BASEBALL (D)	35.00	22.95
☐ 5250 MISSION / THUNDERHEAD (D)	25.00	17.95
☐ 5146 JUPITER MISSION (D)	35.00	22.95
☐ 5252 GULF STRIKE (D)	30.00	19.95
☐ 5254 MACBETH (D)	25.00	17.95
☐ 2375 COMPUTER TITLE BOUT (D)	30.00	19.95
☐ 0860 TOURNAMENT GOLF (D)	29.95	18.95
☐ 5140 BLACK THUNDER (D)	19.95	14.95

Broderbund

☐ 2903 LODE RUNNER (D)	$34.95	$19.95
☐ 2905 KARATEKA (D)	29.95	23.95
☐ 3038 CHAMPION LODE RUNNER (D)	34.95	26.95
☐ 5158 BANK STREET WRITER (D)	49.95	32.95
☐ 5330 BANK STREET SPELLER (D)	49.95	32.95
☐ 5332 BANK STREET FILER (D)	49.95	32.95
☐ 5334 BANK STREET MAILER (D)	49.95	32.95
☐ 2540 PRINT SHOP (D)	44.95	27.95
☐ 2542 GRAPHIC LIBRARY NO. 1 (D)	24.95	15.95
☐ 3898 GRAPHIC LIBRARY NO. 2 (D)	24.95	15.95
☐ 3897 GRAPHIC LIBRARY NO. 3 (D)	24.95	15.95
☐ 2910 PRINT SHOP COMPANION (D)	39.95	24.95
☐ 5160 MUSIC SHOP (D)	44.95	28.95
☐ 5170 LODE RUNNERS RESCUE (D)	29.95	20.95

Electronic Arts

☐ 3830 DR J & LARRY BIRD (D)	$29.95	$23.95
☐ 3832 FINANCIAL COOKBOOK (D)	39.95	27.95
☐ 3834 MAIL ORDER MONSTERS (D)	34.95	22.95
☐ 3840 THE SEVEN CITIES OF GOLD (D)	29.95	23.95
☐ 3842 SKY FOX (D)	29.95	23.95
☐ 5176 CARRIERS AT WAR (D)	42.95	32.95
☐ 5178 REACH FOR THE STARS II (D)	37.95	28.95
☐ 5180 HEART OF AFRICA (D)	29.95	23.95
☐ 5182 MOVIE MAKER (D)	29.95	23.95
☐ 5184 EUROPE ABLAZE (D)	42.95	34.95
☐ 5186 M.U.L.E (D)	19.95	16.95
☐ 5188 MURDER ON ZINDERNEUF (D)	19.95	16.95
☐ 5190 MUSIC CONSTRUCTION SET (D)	19.95	16.95
☐ 5192 PINBALL CONSTRUCTION SET (D)	19.95	16.95
☐ 5194 RACING CONSTRUCTION SET (D)	29.95	22.95
☐ 3601 SUPER BOULDERDASH (D)	29.95	22.95
☐ 3600 TOUCHDOWN FOOTBALL (D)	29.95	22.95

Datasoft

☐ 3025 BRUCE LEE (D)	$34.95	$19.95
☐ 3026 PAC-MAN (D)	34.95	17.95
☐ 3027 MIGHTY CONAN (D)	34.95	22.95
☐ 3028 MR DO! (D)	34.95	18.95
☐ 3029 DIG DUG (D)	34.95	18.95
☐ 3032 POLE POSITION (D)	34.95	18.95
☐ 5218 THE GOONIES (D)	29.95	18.95
☐ 5220 ZORRO (D)	29.95	18.95

Epyx

☐ 0337 WORLD'S GREAT FOOTBALL (D)	$39.95	$23.95
☐ 0338 WINTER GAMES (D)	39.95	20.95
☐ 0339 THE EIDOLON (D)	39.95	20.95
☐ 0340 KORONIS RIFT (D)	39.95	20.95
☐ 0360 JET COMBAT SIMULATION (D)	39.95	20.95
☐ 0364 SUMMER OLYMPIC GAMES (D)	39.95	18.95
☐ 0365 WORLD'S GREAT BASEBALL (D)	34.95	22.95
☐ 0382 SUMMER OLYMPIC GAMES II (D)	39.95	20.95
☐ 0750 PITSTOP II (D)	39.95	22.95
☐ 2046 IMPOSSIBLE MISSION (D)	34.95	16.95
☐ 2066 ROBOTS OF DAWN (D)	39.95	15.95
☐ 2070 BARBIE (D)	39.95	18.95
☐ 2074 G.I. JOE (D)	39.95	18.95
☐ 3005 BALLBLAZER (D)	29.95	20.95
☐ 3006 RESCUE ON FRACTALUS! (D)	29.95	20.95
☐ 1556 MOVIE MONSTER GAME (D)	39.95	24.95
☐ 1557 MICROSOFT MULTIPLAN (D)	59.95	39.95
☐ 1558 PROG BASIC TOOLKIT (D)	44.95	29.95
☐ 1559 VORPAL UTILITY KIT (D)	34.95	22.95

Strategic Simulations, Inc.

☐ 2995 RDF 1985 (D)	$34.95	$20.95
☐ 2997 GEOPOLITIQUE (D)	39.95	23.95
☐ 3008 RINGSIDE SEAT (D)	39.95	23.95
☐ 3010 IMPERIUM GALACTUM (D)	39.95	23.95
☐ 3011 CARTELS AND CUTTHROATS (D)	39.95	23.95
☐ 3012 RAILS WEST (D)	39.95	23.95
☐ 3014 PROFESSIONAL TOUR GOLF (D)	39.95	23.95
☐ 3015 50 MISSION CRUSH (D)	39.95	23.95
☐ 3016 PRESIDENT ELECT (D)	39.95	23.95
☐ 3017 BROADSIDES (D)	39.95	24.95
☐ 3018 COMPUTER QUARTERBACK (D)	39.95	24.95
☐ 3020 COMPUTER AMBUSH (D)	59.95	37.95
☐ 3021 COMPUTER BASEBALL (D)	39.95	23.95
☐ 3031 FIELD OF FIRE (D)	39.95	23.95
☐ 5154 KAMPFGRUPPE (D)	59.95	34.95
☐ 5156 COLONIAL CONQUEST (D)	39.95	23.95
☐ 3768 U.S.A.A.F. (D)	59.95	36.95
☐ 1560 SIX GUN SHOOTOUT (D)	39.95	23.95
☐ 1561 BATTLE OF ANTIETAM (D)	49.95	31.95
☐ 1562 BATTALION COMMANDER (D)	39.95	23.95
☐ 1563 PANZER GRENADIER (D)	39.95	23.95
☐ 1564 NORWAY 1985 (D)	34.95	20.95
☐ 1565 MECH BRIGADE (D)	59.95	36.95
☐ 1567 BATTLEGROUP (D)	59.95	37.95

BUSINESS

Softsync

☐ 5930 ACCOUNTANT, INC. (D) C128	$99.95	$64.95
☐ 5932 PERSONAL ACCOUNTANT (D)	34.95	26.95
☐ 5934 MODEL DIET (D)	29.95	23.95
☐ 5936 TRIO (D) C128	49.95	45.95
☐ 5938 KID PRO QUO (D)	29.95	23.95
☐ 5940 DESK MANAGER (D) C128	39.95	28.95

Timeworks

☐ 0176 INVENTORY MANAGE (D)	$69.95	$38.95
☐ 0180 ACCOUNTS RECEIVABLE/ INVOICING (D)	69.00	38.95
☐ 0182 ACCOUNTS PAYABLE/ CHECKWRITING (D)	69.00	38.95
☐ 0184 PAYROLL MANAGEMENT (D)	69.00	38.95
☐ 0188 GENERAL LEDGER (D)	69.00	38.95
☐ 0928 EVELYN WOOD SPEED READ (D)	69.95	32.95
☐ 5022 WORDWRITER & DATA MANAGER II (D)	98.00	49.00
☐ 5026 SWIFTCALC/SIDEWAYS (D)	49.95	32.95

Business Continued

C128 Software From Timeworks

☐ 5022 WORD WRITER/ SPELL CHECKER (D)	$69.95	$59.95
☐ 5024 DATA MANAGER II (D)	69.95	49.95
☐ 5026 SWIFTCALC WITH SIDEWAYS (D)	69.95	49.95
☐ 5030 PARTNER (D)	59.95	39.95
☐ 3048 SYLVIA PORTER (D)	69.95	39.95

EDUCATION

American Educational Computer

☐ 2482 ELEM. SCIENCE FACTS (D)	$29.95	$14.95
☐ 2492 VOCABULARY WORD BUILD (D)	29.95	14.95
☐ 2493 GRAMMAR WORD SKILLS (D)	29.95	14.95
☐ 2494 WORLD GEOGRAPHY FACTS (D)	29.95	14.95
☐ 2495 SPANISH VOCAB. SKILLS (D)	29.95	14.95
☐ 2496 FRENCH VOCAB. SKILLS (D)	29.95	14.95
☐ 2497 WORLD HISTORY (D)	29.95	14.95
☐ 2498 U.S. HISTORY FACTS (D)	29.95	14.95
☐ 2499 BIOLOGY FACTS (D)	29.95	14.95
☐ 2519 U.S. GEOGRAPHY FACTS (D)	29.95	14.95
☐ 2520 U.S. GOVERNMENT FACTS (D)	29.95	14.95
☐ 2521 AEC SPELLING (D)	39.95	24.95
☐ 3745 PHONICS (D)	39.95	24.95
☐ 3747 LEARN TO READ (D)	39.95	24.95
☐ 3749 READING COMPRENSION (D)	39.95	24.95

Designware

☐ 0824 GRAMMAR EXAMINER (D)	$39.95	$24.95
☐ 0828 SPELLAKAZAM (D)	34.95	9.95
☐ 0832 STATES & TRAITS (D)	44.95	27.95
☐ 0836 SPELLICOPTER (D)	39.95	22.95
☐ 0840 CREATURE CREATOR (D)	34.95	9.95
☐ 0844 TRAP-A-ZOID (D)	39.95	9.95
☐ 2518 THE BODY TRANSPARENT (D)	44.95	27.95
☐ 2517 EUROPEAN NATIONS & LOCATIONS (D)	44.95	19.95
☐ 2062 MATH MAZE (D)	39.95	22.95
☐ 5100 ALGEBRA I (D)	39.95	19.95
☐ 5102 REMEMBER (D)	69.96	49.95
☐ 5104 WEBSTER'S NUMBERS (D)	39.95	19.95
☐ 5105 SPELLING & READ PRIMER (D)	39.95	19.95
☐ 5106 ALGEBRA 2 (D)	39.95	19.95
☐ 5107 ALGEBRA 3 (D)	39.95	19.95

Mindscape

☐ 5108 KEYBOARD CADET (D)	39.95	25.95
☐ 5110 BANK STREET MUSIC WRITER (D)	39.95	25.95
☐ 5112 CROSSWORD MAGIC (D)	49.95	29.95
☐ 5114 THE PERFECT SCORE (D)	69.96	45.95
☐ 5116 COLORME/RAINBOW BRITE (D)	34.95	18.95
☐ 5118 THE HALLEY PROJECT (D)	39.95	22.95
☐ 5120 INDIANA JONES IN THE LOST KINGDOM (D)	29.95	18.95
☐ 5122 BANK STREET STORYBOOK (D)	39.95	22.95
☐ 5910 THE DOLPHIN'S RUNE (D)	29.95	18.95
☐ 5912 THE LUSCHER PROFILE (D)	39.95	22.95
☐ 5914 QUAKE MINUS ONE (D)	29.95	16.95
☐ 5916 THE LORDS OF MIDNIGHT (D)	29.95	16.95
☐ 5918 SHADOWFIRE (D)	29.95	16.95
☐ 3702 BOP 'N WRESTLE (D)	29.95	21.95
☐ 3690 INFILTRATOR (D)	29.95	21.95

Weekly Reader — *Buy 1 Get One Free!*

☐ 2512 STICKYBEAR NUMBERS (D)	34.95	14.95
☐ 2513 STICKYBEAR BASKETBOUNCE (D)	34.95	14.95
☐ 2514 STICKYBEAR OPPOSITES (D)	34.95	14.95
☐ 2515 STICKYBEAR ABC (D)	34.95	14.95
☐ 2516 STICKYBEAR SHAPES (D)	34.95	14.95
☐ 2600 PIC BUILDER (D)	29.95	14.95
☐ 5126 STICKYBEAR SPELLGRABBER (D)	29.95	14.95
☐ 5128 STICKYBEAR TOWN BUILDER (D)	29.95	14.95
☐ 5130 STICKYBEAR MATH (D)	29.95	14.95
☐ 5132 STICKYBEAR READING (D)	29.95	14.95
☐ 5129 STICKYBEAR TYPING (D)	29.95	14.95

Add $3.00 for shipping, handling and insurance. Illinois residents please add 6¼% tax. Add $6.00 for CANADA, PUERTO RICO, HAWAII, ALASKA, APO-FPO orders. Canadian orders must be in U.S. dollars. WE DO NOT EXPORT TO OTHER COUNTRIES, EXCEPT CANADA. Enclose Cashiers Check, Money Order or Personal Check. Allow 14 days for delivery, 2 to 7 days for phone orders, 1 day express mail! Prices & availability subject to change without notice.
VISA — MASTER CARD — C.O.D. No C.O.D. to Canada, APO-FPO

PROTECTO

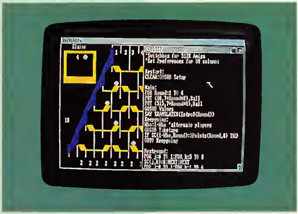
We Love Our Customers

22292 N. Pepper Rd., Barrington, Illinois 60010

312/382-5244 to order

interpret every line of code while the program is running.

An alternative approach is a *compiler*. Popular compiled languages include Pascal, FORTRAN, C, COBOL, and some BASICs. Running a program with a compiler requires two steps. First, the compiler interprets all the commands in the program without carrying them out and creates a new version of the program on disk called *object code, p-code,* or *run-time code*. This file, incomprehensible to human eyes, is quite similar to a program written



In Amiga BASIC line numbers are unnecessary, and are replaced with labels that indicate subroutines and program divisions.

in machine language—it's a complex pattern of bits which is the only language that any computer really understands. After the compilation is completed, the object code can be run. Since all the commands in the program have already been interpreted by the compiler, the object code runs much faster than a program which must be interpreted one command at a time.

Unfortunately, the two-step process of compiling can take many minutes, which is frustrating for programmers—especially beginners. If the program contains an error, the whole process has to be repeated. It's not as frustrating as the old batch processing, but it's a step back in that direction.

Compiled languages also require more computer power than interpreted languages. They need faster processors and more memory—sometimes 512K of Random Access Memory (RAM) is scarcely enough for a compiler. This has largely prevented compilers from becoming popular on personal computers, since until recently most machines were limited to 64K of RAM. Because BASIC interpreters can be squeezed into as little as 8K of RAM, BASIC was the logical choice for the first generation of microcomputers.

Most personal computer owners who are interested in programming quickly grow accustomed to the version of BASIC that comes with their machines. Some BASICs are built into the computer's Read Only Memory (ROM), while others are supplied on plug-in ROM cartridges or floppy disks. But there have been significant variations of BASIC even for the same brands of computers. Besides that, additional versions of BASIC are often made available by independent sources, as are packages which add enhancements to existing BASICs.

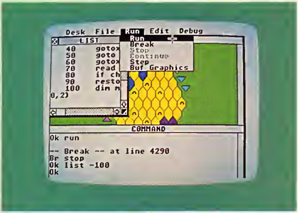
For example, Applesoft BASIC is a version of Microsoft BASIC that's used by Apple II-series computers. Integer BASIC, an earlier BASIC from Apple, was available in ROM on the original Apple II. Although faster than Applesoft BASIC, Integer BASIC doesn't allow floating-point math operations (the use of fractions) as does Applesoft. The Apple II+ machine came with Applesoft BASIC in ROM, while the Apple IIe and IIc computers have Applesoft BASIC on built-in language cards. (The II+ can add a language card, too.)

Atari computers have had several different versions of BASIC available, as well as optional third-party BASIC languages. The Atari 400, 800, and 1200XL computers come with an 8K BASIC ROM cartridge, while the 600XL, 800XL, and 130XE computers have later revisions of BASIC built into ROM. There are also alternatives to Atari BASIC, such as Microsoft BASIC from Atari and BASIC XL and BASIC A+ from Optimized Systems Software (OSS) in San Jose, California.

Commodore 64 owners are familiar with the BASIC 2.0 version in their computers, the same version that appeared in the earlier VIC-20. Prior to 2.0, the earlier Commodore PET computer included a version 4.0. A variation

between BASIC 2.0 and BASIC 4.0, called BASIC 3.5, was included in the Plus/4 and 16 computers. And the 128 includes a powerful version of BASIC, 7.0, that contains virtually all of the commands of the earlier BASICs.

But despite the differences among these forms of BASIC, they're all fairly similar in their organization. They're interpreted, giving immediate feedback to the user; they all use similar *commands*, *variables*, and *functions*; each line of BASIC begins with a line number;



ST BASIC has line numbers, but the GEM operating environment includes multiple windows, drop-down menus, and icons much like AmigaBASIC and Macintosh BASIC.

BASIC doesn't usually permit your computer to crash, so it's friendly to programmers; and access to printers and other peripherals is relatively easy to accomplish.

Within the past year, as the new Commodore Amiga and Atari ST computers have joined the Apple Macintosh—machines based on the more powerful 68000 microprocessor—computer users have been confronted with new BASIC languages that have several important differences from earlier versions. The Macintosh and the Amiga have BASIC languages that are almost identical, both created by Microsoft. The Amiga was initially released with a BASIC language called ABASIC, but that was superseded by the Microsoft version, called Amiga BASIC. The Atari ST, at this writing, has an ST BASIC from Atari, as well as several other versions of BASIC that should be available from third-party companies by the time you read this.

Both Amiga BASIC and Macintosh BASIC abandon the line numbers used in previous BASICs. Instead, meaningful labels are used that identify sections of code and subroutines. Although ST BASIC does have line numbers, you can put labels within lines and direct subroutines to those labels.

The programming environment changes as well. Windows—with separate areas for your commands, the program listing, and the program output—take the place of the single screen you may be used to. With this system, you can actually see the program run while the program's code stays visible. Using a mouse, you can click on menu items like RUN and LIST instead of typing them in.

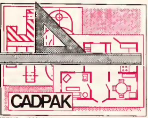
Macintosh BASIC, for example, offers several windows: a Command window to take your directions; two List windows, allowing you to have two different parts of the program onscreen at the same time; and an Output window which allows you to see the results of your programming. There are also programming tools that simplify your efforts. TRACE MODE, a debugging tool that can be switched on or off, highlights whatever line in your program is currently executing. The new BASICs also generally support the currently popular mouse environment, allowing you to create your own custom-designed windows, pull-down menus, and dialog boxes.

ST BASIC is fundamentally similar to the BASICs you may have used on your eight-bit computer, but offers accessibility to windows, drop-down menus, and graphic icons from the GEM Desktop environment much like Amiga BASIC and Macintosh BASIC. There are actually four windows on the ST BASIC screen: Output, List, Command, and, hidden behind the first three, an Edit window.

It would, of course, be precipitate to conclude that the new BASICs represent the ultimate in man-machine communication. Rather, they seek to offer a higher level of power, ease, and efficiency to the computer programmer. Computer languages are continually evolving as the search continues for ever more effective methods by which man can interact with his increasingly intelligent inventions. ©

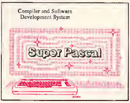
'128™ and C-64™ SENSATIONAL SOFTWARE

The *complete* compiler and development package. Speed up your programs 5x to 35x. Many options: flexible memory management; choice of compiling to machine code, compact p-code or both. '128 version: 40 or 80 column monitor output and FAST-mode operation. '128 Compiler's extensive 80-page programmer's guide covers compiler directives and options, two levels of optimization, memory usage, I/O handling, 80 column hi-res graphics, faster, higher precision math functions, speed and space saving tips, more. A great package that no software library should be without. **128 Compiler $59.95**
64 Compiler $39.95

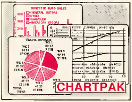


Remarkably easy-to-use interactive drawing package for accurate graphic designs. New dimensioning features to create exact scaled output to all major dot-matrix printers. Enhanced version allows you to input via keyboard or high quality lightpen. Two graphic screens for COPYing from one to the other. DRAW, LINE, BOX, CIRCLE, ARC, ELLIPSE available. FILL objects with preselected PATTERNS; add TEXT; SAVE and RECALL designs to/from disk. Define your own library of symbols/objects with the easy-to-use OBJECT MANAGEMENT SYSTEM–store up to 104 separate objects.
C-128 $59.95
C-64 $39.95

For school or software development. Learn C on your Commodore with our in-depth tutorial. Compile C programs into *fast* machine language. C-128 version has added features: Unix™-like operating system; 60K RAM disk for fast editing and compiling Linker combines up to 10 modules; Combine M/L and C using CALL; 51K available for object code; Fast loading (8 sec. 1571, 18 sec. 1541); Two standard I/O librarys plus two additional libraries—math functions (sin, cos, sqrt, etc.) & 20+ graphic commands (line, fill, dot, etc.).
C-128 $79.95
C-64 $59.95



Not just a compiler, but a complete system for developing applications in Pascal with graphics and sound features. Extensive editor with search, replace, auto, renumber, etc. Standard J & W compiler that generates fast machine code. If you want to learn Pascal or to develop software using the best tools available–**SUPER Pascal** is your first choice.
C-128 $59.95
C-64 $59.95



Easily create professional high quality charts and graphs without programming. You can immediately change the scaling, labeling, axis, bar filling, etc. to suit your needs. Accepts data from CalcResult and MultiPlan. C-128 version has 3X the resolution of the '64 version. Outputs to most printers.
C-128 $39.95
C-64 $39.95

PowerPlan

One of the most powerful spreadsheets with integrated graphics. Includes menu or keyword selections, online help screens, field protection, windowing, trig functions and more. PowerGraph, the graphics package, is included to create integrated graphs and charts. **C-64 $39.95**

Technical Analysis System for the C-64	$59.95
Ada Compiler for the C-64	$39.95
VideoBasic Language for the C-64	$39.95

OTHER TITLES AVAILABLE:

COBOL Compiler

Now you can learn COBOL, the most widely used commercial programing language, and learn COBOL on your 64. COBOL is easy to learn because its easy to read. COBOL Compiler package comes complete with Editor, Compiler, Interpreter and Symbolic Debugger. **C-64 $39.95**

Personel Portfolio Manager

Complete portfolio management system for the individual or professional investor. Easily manage your portfolios, obtain up-to-the-minute quotes and news, and perform selected analysis. Enter quotes manually or automatically through Warner Computer Systems. **C-64 $39.95**

Xper

XPER is the first *"expert system"* for the C-128 and C-64. While ordinary data base systems are good for reproducing facts, XPER can derive knowledge from a mountain of facts and help you make expert decisions. Large capacity. Complete with editing and reporting. **C-64 $59.95**

C-128 and C-64 are trademarks of Commodore Business Machines Inc
Unix is a trademark of Bell Laboratories

Abacus Software

P.O. Box 7219 Dept. C6 Grand Rapids, MI 49510 - Telex 709-101 - Phone (616) 241-5510

Call **now** for the name of your nearest dealer. Or to order directly by credit card, MC, AMEX of VISA call **(616) 241-5510**. Other software and books are available–Call and ask for your **free** catalog. Add $4.00 for shipping per order. Foreign orders add $12.00 per item. Dealer inquires welcome–1400+ nationwide.

C and the 68000

Kathy Yakal, Assistant Features Editor

A programming language written almost 15 years ago for a specific purpose—to develop an operating system for a mainframe computer—has been getting a lot of attention in the microcomputer community lately. C, a language described by many as simple, elegant, and powerful, is especially well-suited for programming the 68000 chip housed in the Commodore Amiga, Atari ST, and Apple Macintosh.

C is a beguiling language. In structure and difficulty, it falls somewhere between machine language and a higher level language such as BASIC or Pascal. C was developed in 1972 by Dennis Ritchie, who wrote it specifically to design the UNIX operating system running on the PDP-11 (a mainframe computer). Though some mainframe and minicomputer programmers have chosen to use it in the years since, it's enjoying a renewed popularity with the advent of the new 68000-based machines: the Atari ST, the Apple Macintosh, and the Commodore Amiga.

C is the language of choice for the Amiga and the ST, say many programmers, for four main reasons. First, its basic command structure is quite concise, but can be extended by individual programmers for specific functions. Second, its relative closeness to actual machine language gives the programmer tremendous power. Third, there is a harmony between the 68000 microprocessor and the C language; 68000 machine language itself supports C constructs, thanks in part to the similarities between the PDP-11 and the 68000. Finally, and perhaps most important for the home computer market, programs written in C on one 68000-based machine can be more easily and quickly transported to another computer than can programs written in many other languages.

Programmers consider several factors when deciding what language to use. Of course, everyone has favorites based on personal experience, but the physical capabilities of individual computers and the type of application being written necessarily create some restrictions.

Just as some computers are designed to support specific lan-

guages, some languages have been developed to support specific applications. You *could* accomplish almost anything in almost any language, but there is significant variability between languages in the efficiency of program writing and excecuting. Some languages are simply more appropriate than others for specific jobs.

One of the first high-level languages, FORTRAN, was written in 1954, and is geared especially for scientific formula translation. COBOL, developed around the same time, best supports business applications. BASIC and Pascal were intended to be teaching languages, sometimes making them unwieldy for particular applications.

C, written to develop an entire operating system, is less application- and machine-specific, and is amenable to various kinds of programs. Its skeletal architecture allows programmers—once they've learned the sparse command structure—to add their own routines, commands, and input/output functions called *libraries* (standard C libraries are also available commercially). It's really like the construction of an onion: You have a tiny seed in the center and layer upon layer covering it up. And the fact that input/output is extrinsic to the C language makes for greater portability.

So even though the language itself is small, you can end up with very large programs if you don't economize on your use of libraries, warns Tom Hospelhorn, senior analyst at Mindscape Software. "Sometimes you have very small C programs compiling to what seems to be a very large object," he says. "But what that's usually caused by is you've included a standard library of input/output functions with your object code even though your program may not actually use those." So if you're serious about keeping your finished C programs small, you want to avoid the automatic inclusion of a standard library. You can exclude any unused functions.

C's affinity with machine language gives the user greater programming power, but can also create big problems for novice programmers. Some programmers suggest that C is best not attempted by

A Sample Of C

Here is an example program which prints the numbers from one to ten and describes each number as odd or even. Program 1 is written in Amiga BASIC. By comparing it to Program 2, the C version, you can get a sense of some of the differences between the two languages. For one thing, C makes liberal use of braces—{ }—to group statements into a compound statement, or block, which the language treats as a single statement. Functions are called by supplying the name of the function, with arguments in parentheses—the entire program is a function called main( ). Variables are declared before use, and comments are framed with /* and */ characters.

Program 3 is C in a somewhat more condensed, but less easily visualized format. The IF-ELSE construct has been collapsed into one line using the conditional operator (? :) to designate the alternative results of the test. This brief sample, however, can't do justice to the qualities which make C an increasingly popular language. There are several excellent texts on C for beginners, available in most bookstores, which will give you a sense of the language's flexibility, power, and efficiency.

Program 1:

```
REM This is a demo program written in AmigaBASIC

PRINT "The numbers from 1 to 10:"

FOR count = 1 TO 10
  PRINT count;"  ";
  IF (count AND 1) = 1 THEN PRINT "odd" ELSE PRINT "even"
  NEXT count
```

Program 2:

```
/* This is a simple demonstration of a C program,
   written for the Lattice C compiler on the Amiga
   computer */

main()
{
  int count;

  printf("The numbers from 1 to 10:\n");

  for (count = 1;count <= 10;count++)
  {
    printf(" %d  ",count);

    if (count & 1 == 1)
      printf("odd\n");
    else
      printf("even\n");
  }
}
```

Program 3:

```
/* A more compact C version of the demo program */
main()
{
  int count;

  printf("The numbers from 1 to 10:\n");
  for (count = 0;++count <= 10;
    printf(" %d  %s\n",count,count & 1 ? "odd" : "even"));
}
```

COMPUTER MAIL ORDER

THE CMO ADVANTAGE

TO ORDER
CALL TOLL FREE
1-800-233-8950
DEPARTMENT A207

OR MAIL YOUR ORDER TO:

COMPUTER MAIL ORDER
Department A207
477 E. Third Street
Williamsport, PA 17701

AMERICAN EXPRESS · VISA · MasterCard · DINERS CLUB

POLICY

Add 3% (Minimum $7.00) shipping and handling. Larger shipments may require additional charges. Personal and company checks require 3 weeks to clear. For faster delivery use your credit card or send cashier's check or bank money order. Pennsylvania residents add 6% sales tax. All prices are subject to change and all items are subject to availability. Defective software will be replaced with the same item only. Hardware will be repaired or replaced at our discretion within the the terms and limits of the manufacturer's warranty. We cannot guarantee compatibility. All sales are final and returned shipments are subject to a restocking fee.

EDUCATIONAL INSTITUTIONS
CALL TOLL FREE
1-800-221-4283

CUSTOMER SERVICE
& TECHNICAL SUPPORT
1-717-327-1450

CANADIAN ORDERS
1-800-268-3974
Ontario/Quebec

1-416-828-0866
In Toronto

1-800-268-4559
Other Provinces

TELEX: 06-218960

2505 Dunwin Drive,
Mississauga, Ontario
Canada L5L1T1

All prices shown are for U.S.A. orders. Call the Canadian Office for Canadian prices.

THE CMO ADVANTAGE

- Next day shipping on all in-stock items
- Free easy access order inquiry
- Orders from outside Pennsylvania save state sales tax
- Free technical support from our factory trained technicians
- There is no limit and no deposit on C.O.D. orders
- There is no extra charge for using your Visa or MasterCard and your card is not charged until we ship
- No waiting period for cashier's checks
- We accept purchase orders from qualified corporations. Subject to approval
- Educational discounts available to qualified institutions. (See the toll free educational phone number above.)
- FREE CATALOG MEMBERSHIP

HOME COMPUTERS

ATARI

65XE (64K)	**$89.99**
130XE (128K)	**$139.00**
520ST (512K)	**$399.00**

520ST Monochrome System
- 520ST with modulator
- disk drive
- mouse
- logo
- Basic
- 1st Word
- monochrome monitor

LOW, LOW SYSTEM PRICE **$649.00**

520ST Color System
- 520ST with modulator
- disk drive
- mouse
- logo
- Basic
- 1st Word
- color monitor

LOW, LOW SYSTEM PRICE **$799.00**

600XL 64K	CALL
1010 Recorder	$49.99
1050 Disk Drive	$149.00
1020 Printer	$29.99
1027 Letter Quality Printer	$129.00
1030 Direct Connect Modem	$59.99
Comrex 220 Atari	$69.99

APPLE

APPLE IIe	CALL
APPLE IIc	CALL
IIc LCD Display	$329.00

COMMODORE

Amiga Package
- 512K
- 2 Drive
- RGB Monitor — **$1599.00**

C64 Package
- C64
- C1541
- Taxan 220 — **$495.00**

C128 Package
- C128
- C1571
- NAP8562 Monitor — **$775.00**

C128 Computer	**$255.00**
C1571 (Disk Drive for C128)	**$249.00**
C1902 (RGB 13" Monitor for C128	**CALL**
C1670 (Modem for C128)	**$179.00**
C1530 Datasette	$39.99
C1660 Auto Modem	$59.99
DPS 1101 Daisy Printer	$339.00
Comrex 220 (C64 Interface)	$69.99
Xetec SuperGraphix 8K	$69.99

PORTABLE COMPUTERS

NEC

PC-8401 LS	$699.00
PC-8201 Portable Computer	$339.00
PC-8231 Disk Drive	$599.00
PC-8221A Thermal Printers	$149.00
PC-8281A Data Recorder	$99.99
PC-8201-06 8K RAM	$79.99

SHARP

PC-1350	$149.00
PC-1261	$149.00
PC-1500A	$169.00
PC-1250A	$89.99
CE-125 Printer/Cassette	$129.00
CE-150 Color Printer Cassette	$149.00
CE-161 16K RAM	$129.00

HEWLETT PACKARD

41CV	$139.00
41CX	$199.00
HP 11C	$49.99
HP 12C	$75.99
HP 15C	$75.99
HP 16C	$89.99
HPIL Module	$96.99
HPIL Cassette or Printer	$359.99
Card Reader	$143.99
Extended Function Module	$63.99
Time Module	$63.99

We stock the full line of HP calculator products

MODEMS

ANCHOR

Volksmodem	$59.99
Volksmodem 300/1200	$169.00
Signalman Express	$259.00
Lighting 2400 Baud	$399.00
Expressi (PC Halfcard)	$169.00
6470 (64/128) 300/1200 Baud	$139.00

AST

Reach 1200 Baud Half Card	$399.00

DIGITAL DEVICES

AT300 - 300 Baud (Atari)	$99.99

EVEREX

1200 Baud Internal (IBM/PC)	$179.00

Hayes

Smartmodem 300	$139.00
Smartmodem 1200	$369.00
Smartmodem 1200B	$359.00
Smartmodem 2400	$599.00
Micromodem IIe	$149.00
Smart Com II	$69.99
Chronograph	$199.00
Transet 1000	$309.00

Novation

Smart Cat Plus	$299.00
J-Cat	$99.99
Novation 2400	$499.00
Apple Cat II	$219.00
212 Apple Cat II	$379.00
Apple Cat 212 Upgrade	$229.00

QUADRAM

Quadmodem II	
300/1200	$339.00
300/1200/2400	$499.00

SUPRA

MPP-1064 AD/AA (C-64)	$69.99

ACCESSORIES

AMARAY

80 Column Printer Stand	$14.99

CURTIS

Side Mount SS-1	$19.99
Side Mount AT SS-2	$34.99
Universal Stand SS-3	$19.99
Diamond SP-1	$29.99
Emerald SP-2	$39.99
Sapphire SPF-1	$49.99
Ruby SPF-2	$59.99

DATA SHIELD

300 Watt Backup	$379.00
500 Watt Backup	$569.00
Turbo 350 Watt Backup	$449.00
P125 Power Director	$99.99
P150 Power Director w/Modem	$119.00

KENSINGTON

Master Piece	$99.99
Master Piece +	$119.00

KEYTRONICS

KB5150/KB5151/KB5151Jr	CALL
KB5152B/KB5153/KB5149Jr	CALL

MEMORY CHIPS

4164 RAM Chips	(ea.) $1.99
128 RAM Chips	(ea.) $12.99
256 RAM Chips	(ea.) $10.99

Polaroid

Palette	$1399.00
Power Processor	$229.00
Illuminated Slide Mounter	$39.99
Polacolor 2 Pack film	$16.99

DISKETTES

Dennison

Elephant 5¼" SS/SD	$9.99
Elephant 5¼" SS/DD	$11.99
Elephant 5¼" DS/DD	$14.99
Elephant Premium DS/DD(50)	$69.99
Elephant 3½" SS/DD	$24.99

IBM

5¼" DS/DD floppy disks (Box of 10)	$26.99

DENERIC

DS/DD w/Flip'n'File 10	$11.99

maxell

3½" SS/DD (10)	$18.99
3½/II 5 pack SS/DD/Case	$9.99
3½" DS/DD (10)	$29.99
5¼" MD-1 SS/SD (10)	$11.99
5¼" MD-2 DS/DD (10)	$16.99
5¼" MD-2-HD for AT (10)	$29.99

Verbatim

5¼" SS/DD	$12.99
5¼" DS/DD	$24.99
Disk Analyzer	$24.99

DISK HOLDERS

AMARAY

50 Disk Tub 5¼"	$9.99
30 Disk Tub 3½"	$9.99

INNOVATIVE CONCEPTS

Flip'n File 10	$2.49
Flip'n File 50	$14.99
Flip'n File 50 w/lock	$19.99
Flip'n File Data Case	$9.99

DRIVES

HARD

CORE	
AT20-AT72MB	CALL
EVEREX	
60 Meg Internal Backup System	$799.00
IDMEDA	
A110H Single 10	CALL
A210H 10 + 10	CALL
A120H Single 20	CALL
A220H 20 + 20	CALL
Save on 10 & 20 Carts	CALL
IRWIN	
Tape Backup	CALL
KITS	
10 Meg with controller	$399.00
20 Meg with controller	$499.00
PRIAM	
40, 60 MB Inner Space	CALL
Shared Data	CALL
Shared Space	CALL
TALLGRASS	
25, 35, 50, 80 meg (PC)	from $1299.00

FLOPPY

ALLIED TECHNOLOGY	
Apple II, II+, IIe ½ height	$109.00
INDUS	
Atari GT	$199.00
C-64 /128 GT	$199.00
MSD	
SD1 C-64 Single	$219.00
SD2 C-64 Dual	$469.00
TANDON	
320K 5¼" (PC)	$119.00
TEAC	
320K 5¼"	$119.00

DEPT. A207 CALL TOLL-FREE

people unfamiliar with machine language. At the very least, a disciplined, structured approach to C programming is highly recommended.

"If you haven't done a lot of assembly language first, going from Pascal or BASIC to C is going to be more of a challenge because it is getting closer to the machine," says Jeff Steinwedel, engineering manager at Activision Software. "It's like going from BASIC to assembly language. You have to know a lot more about what's going on internally. The language system is going to be less help in bailing you out of a situation or preventing you from making bugs in the first place. But if you've done a lot of assembly language programming, the transition to C will be fairly straightforward."

One of the characteristics shared by C and machine language is the ability to manipulate memory directly (in C, *pointers* are used). In higher level languages, memory access is often indirect. And C's heavy reliance on pointers is very powerful in general, says Hospelhorn.

But because you tend to do a lot of things with pointers, and because there's no error checking on pointers to see what they point to, you can wind up creating problems for yourself if you're not a disciplined programmer, he says. "It's possible to damage your program, to set memory locations that you didn't intend to if you're not well aware of where your pointers are pointing. That can make debugging sort of tedious. It's certainly possible for a programmer to generate very difficult errors—more so than some other languages that do a better job of checking things for you."

C, then, isn't a free lunch. What C gives you is a tremendous amount of power and flexibility, a feeling of liberation, he says, because most of the things you can think of, you can do in some way. But with that liberation comes a certain amount of responsibility. "It's always best to plan things out before forging ahead, but it might be a little more true in C."

And there's a big payoff after a C program is successfully completed and debugged: Translating that program to run on another 68000-based computer is easier than transporting a program from, say, an Apple II to an Atari 800. Part of the reason for this, of course, lies in the memory limitations of the eight-bit machines. "When you're in a very constrained machine without a lot of memory, like the Apple II, you tend to do a lot more in assembly language because you don't have the space," says Steinwedel. "With an Atari 1040 ST with a megabyte of memory, it probably isn't going to hurt you.

"But the real advantage is to be able to produce the code quickly and rapidly transport it from one environment to another. Something running in GEM on an IBM can move very quickly to the ST."

There's one additional reason for the popularity of C on the 68000-based machines. It's often mentioned almost as an afterthought, but it's significant. There are currently many good C compilers available. Other languages aren't as well-supported at this time with efficient, tested compilers. Lattice (Glen Ellyn, IL) and Manx Software Systems (Shrewsbury, NJ) publish C compilers that are enjoying popularity with 68000-based computer programmers, as do several other software publishers. There are also versions of interpreted C available, for programmers who want to debug more easily. Some use interpreters to write the programs and then, after all the problems are ironed out, run the source code through a compiler for the greatest runtime efficiency.

Although the friendly user interfaces of the Apple Macintosh, Atari ST, and Commodore Amiga have enticed nontechnical consumers to purchase them as applications machines, sales of languages and how-to books and other programming utilities indicate that many buyers are planning to write their own programs. The special benefits of C on 68000 machines recommend it to experienced machine language programmers. Likewise, people who have worked with higher level languages and want to move a bit closer to the inner workings of the computer might want to sample this special way of communicating with their machines. A decade and a half after its inception, the C programming language is approaching a new level of popularity. ©

The **Top Five** Free Programs For Your Computer

Arlan R. Levitan

Good software doesn't have to be expensive. You can accumulate a respectable software library merely by taking advantage of the thousands of programs in the public domain—that is, programs which are given away free by their authors. Another alternative, the "shareware" concept, lets you test-drive a program for free and make a voluntary contribution if you like. Here's a guide to public domain software and shareware, plus the results of a survey in which users all over the country voted for their top five favorites.

Does the thought of paying more for a program than you laid out for your computer make you grumpy and irascible? Cheer up. There's a wealth of programs available for your computer that cost little or nothing at all. Public domain and shareware programs can provide you with a never-ending supply of grist for your computer's mill.

The idea of public domain software has been around since the early computer hobbyists first started sharing their programs with each other. People would try running each other's programs, suggest improvements, or make the improvements themselves. Few people copyrighted their programs because they were hobbyists rather than software authors trying to make a living. Legally, all it takes to place a program in the public domain is for the author to declare it so. (Of course, this excludes most programs published in magazines and books, which are nearly always copyrighted to protect the authors.)

Public domain programs can be freely exchanged between individuals or distributed by user groups and computer bulletin board systems (BBSs). They come with no warranties, packaging, or customer support. They are gifts to the public and vary in quality from marginal to very good.

To determine which public domain programs are the most popular among users, in April we conducted a survey over three commercial information services: CompuServe, The Source, and Delphi. Below are the results of this informal survey. For each personal computer, we've listed the top five programs. The type of program is identified within parentheses.

We have excluded from consideration programs that are not truly in the public domain, including programs which elicit a fee for documentation, and programs which have been published, are in widespread use, but are definitely not in the public domain—such as COMPUTE!'s own *SpeedScript*, for example.

You'll notice that many of the popular programs on the list are terminal programs. This is probably due to the fact that the survey was conducted online among telecomputing enthusiasts.

To obtain copies of any of these programs, try contacting your local user group or logging onto a BBS or commercial information ser-

vice. Friends and coworkers are also valuable sources for public domain programs.

Another type of freely distributed software that is sometimes confused with public domain software is *shareware* (also called *user-supported software*). The concept of shareware came about as a response to the negative aspects of marketing software commercially.

It seems that almost everybody likes to complain that software is too expensive. Critics of the software industry claim that prices are inflated by a charge-what-the-market-will-bear attitude as the product filters through distribution channels. The manufacturer typically sells to a distributor, who in turn sells to a retailer. Each middleman adds a markup. The author of the software receives only a small percentage of the selling price.

Critics argue that this practice causes a serious problem: The perception of high prices encourages unauthorized duplication of software. This leads to a classic conflict between the manufacturers and the software pirates. Manufacturers may be tempted to boost their prices to make up for expected losses to piracy, and pirates may justify copying because they say prices are unreasonably high.

For these and other reasons, some software authors decide to market their programs themselves. There have been few success stories among those who've tried this approach. The authors attempt to work within the established marketplace, but usually fail because they lack the resources necessary to promote, advertise, and distribute their product.

About four years ago, a programmer named Andrew Fluegelman wrote a terminal program for IBM computers called *PC-Talk III*. To distribute his program, Fluegelman combined aspects of both public domain and commercial software to come up with a new category he called *Freeware*. Freeware is based on three concepts:

• Before buying a program, computer users should have the opportunity to fully assess its value by using it extensively to determine whether it serves their needs.

• Original software of high quality written by independent authors will be supported by the personal computing community.

• Copying of these programs should be *encouraged*, rather than discouraged. The ease of disseminating programs outside traditional commercial channels should be exploited by software authors to maximize distribution.

Fluegelman actually trademarked the term Freeware, so as these ideas spread and other authors began following suit, the term *shareware* was coined for general use. Here's how shareware typically works:

Anyone can get a copy of a shareware program. Usually, you obtain it from a local user group or BBS. Since there is no packaging or manual, any documentation is generally in the form of a text file on the disk or BBS. You must print out a hardcopy if you want a manual for reference purposes.

Shareware programs contain a notice suggesting that you send a certain contribution to the author if you find the program useful. The contribution is voluntary, and even if none is made, you're encouraged to share the program with others.

Although no shareware authors are reported to be making a killing, many are said to be realizing a steady stream of supplemental income.

How good is shareware? The best of it is quite good indeed, and often better suited to the needs and abilities of casual users than more expensive commercial programs. If you're willing to do without fancy manuals and can rely on fellow users for technical support, shareware may be right for you.

Here are the top five freely distributed programs for each popular personal computer. Shareware programs are denoted with an asterisk (*). You'll notice that only four programs are listed for the Commodore 64/128. That's because the other programs which received votes are not truly in the public domain—including two which are copyrighted by COMPUTE!.

Commodore 64/128
Comm Term (Terminal program)
*Haunted Hill (Game)**
Disk Doctor 128 (Utility)
Blue Thunder (Game)

Atari 400/800/XL/XE
AMIS (Bulletin board system)
AMODEM (Terminal program)
MYRIAPEDE (Game)
POKEY Player (Music)
AMENU (Program autoloader)

Atari ST Series
STerminal (Terminal program)
*STCalc (Calculator desk accessory)**
Megaroids (Game)
RMDISK (RAM disk utility)
COPY (File utility for single-drive systems)

Apple II Series
EAMON (Adventure game)
*FreeWriter (Word processor)**
*EVE (Terminal program)**
RAMDISK128 (RAM disk utility)
ABBS (Bulletin board system)

Commodore Amiga
Aterm (Terminal program)
StarTerm (Terminal program)
Mandelbrot (Graphics demo)
Hack (Adventure game)
EMACS (Text editor)

IBM PC/PCjr
MEMBRAIN (RAM disk utility)
*PROCOMM (Terminal program)**
*PC-File (Database manager)**
RBBS (Bulletin board system)
*PC-Write (Word processor)**

Apple Macintosh
*Red Ryder (Terminal program)**
*BINHEX (File conversion utility)**
MazeWars (Game)
VMCO (Vocal/visual terminal program)
ResEdit (Resource editor)

Texas Instruments TI-99/4A
Fast-Term (Terminal program)
Disk Manager 1000 (Disk cataloger)
FUNL Writer (Word processor)
NeatList (Utility)
MassCopy (Utility)

©

Hex War

Todd Heimarck, Assistant Editor

You float high above a distant planet, controlling robot armies below. Can you take control of the priceless mining turf planetside, or will your opponent's robot crews prevail? To win at this thoughtfully designed, engaging strategy game, you'll need foresight and conceptual skills rather than a quick hand on the joystick. The original version is written for the Commodore 128. We've programmed new versions for the Commodore 64, Apple II series, IBM PC/PCjr, Atari 400/800/XL/XE, and the Amiga. A joystick is required to play the Commodore 128 and 64 versions. The IBM PC/PCjr version requires Cartridge BASIC for the PCjr and BASICA plus a color/graphics adapter for the PC. The Atari version requires a joystick and at least 48K of memory. The Amiga version requires 512K.

"Hex War" is a two-player strategy game that can be played five different ways, and there are limitless variations. But the basic premise is always the same: You and an opponent move armies on a field of hexagons, attempting to capture territory.

The goal of the first two games is simple: capture the capital city of the other player. In game 1, the capital cities are far apart; you must devote some of your armies to defending your own capital while attempting to breach the walls of the other capital. Game 2 puts the capitals near each other, so offense and defense tend to merge in this scenario. Most of the action takes place within a small area of the battlefield.

Games 3 and 4 spread the action over a wider area. In the third game, your object is to occupy eight of the twelve cities on the game board. Six cities occupy the periphery, and six are in the center of the playfield. Game 4 requires actual control of six cities; you must have an army in the city, one that's not involved in a battle, before you're credited with control (this version will probably take the most amount of time to play).

Although the first four scenarios encourage a commitment to battle, you employ different tactics in the fifth. The goal here is to acquire 40 of the 61 hexes, so you need some free armies to move around. As soon as you claim 40 hexes, you win the game.

Typing It In

Hex War is written in BASIC, with some important information in DATA statements. Type in the version for your computer and be sure

to save a copy. Refer to the notes below for special instructions specific to your computer. After the game has been saved, type RUN to begin playing.

When you first run Hex War, the computer pauses to set up the screen, then displays a menu of five choices. The five different games are explained in detail below. If you're new to the game, press the 1 key to choose game 1. There will be another short pause while the variables are initialized, and then you'll see a playfield with 61 hex shapes, containing four armies on each side (see photos).

Hexes And Hexadecimal

A chess board has 64 squares arranged in a rectilinear grid. Hex War gives you a playing field of 61 hexagons (almost as many as a chess board), but they're part of a six-sided honeycomb field. If you've played war games before, you may recognize the hexes.

If you're using a Commodore 128, 64, or Atari computer, plug in the joystick before playing (use port 2 for the 128 and 64). The other versions use keyboard controls as explained below. At first, the cursor movement may seem unusual. The cursor travels not up-down/left-right, but northeast-southeast/northwest-southwest. To make the movement less confusing, turn your joystick 45 degrees clockwise, so that what was up becomes northeast, and so on.

Each hex has six neighbors, so an army can move in six possible directions. To travel left and right, you'll have to push the joystick twice (for example, up and right on the joystick to move one hex to the right, which counts as one movement).

Army strengths are listed in hexadecimal (base 16) numbers, so the four armies labeled 40 actually have strengths of 64 (the hexadecimal value 40 equals 64 in our everyday decimal numbering system). At the beginning of a turn, any army has exactly three movement points. It requires one point to move an army into a neutral or enemy-controlled zone. To move *through* the same zone also requires a point. To move into and through a friendly hex requires a total of one point. This means you can move a single army through two neutral or enemy hexes in any one turn, but the same army can move through up to three friendly zones during a turn.

Select an army by moving the cursor onto it. Click the joystick button once, then position the cursor on a neighboring hex and click again. If you wish to stop, click again, and two plus signs (++) will appear, signaling that no more movement can occur. Otherwise, position the cursor on another neighboring hex and click.

Zones Of Control

Each army controls the six contiguous hexes surrounding its resident hex. If you enter an enemy's zone of control, you forfeit any additional moves and must prepare for battle. In addition, an army that begins the turn in a zone of control cannot move until the battle is resolved.

Robots Vs. Robots

In this game, you aren't really on the planet, but parked high above it in a remote mothership. You've landed some robots to explore the area, and they've encountered robots belonging to another explorer. Your robots, or *bots* as you call them, follow your orders to advance toward the other bots. Each bot has a mining laser which can stop or disable the other bots. Also, your bots have disruptor beams which can daze another bot, temporarily confusing it. When two bot-groups come close to each other, they shoot lasers and disruptors until one army of bots is disabled.

Three things can happen to a robot which suffers a hit. If the robot suffers a direct hit in its logic unit by a laser, it is vaporized. It is destroyed forever and never reappears in play.

The second thing that can happen is injury. If the laser beam is deflected, the robot is out of commission until it can be transported back to a botspital. An injured bot is frozen in place until the battle is finished, after which the victorious army carts away the injured bots to be repaired and reused.

Thus, winning a battle means you evacuate both the friendly injured and the enemy injured. After all of the injured bots recover, *they join the force in whose botspital they were healed.* In effect, injured bots eventually become members of the army which won the battle in which they were damaged.

The third possibility is confusion: The robot is temporarily disoriented for two turns. When the time has passed, the robot is ready again.

Reprogramming Bots

Moving the cursor onto an army of robots brings up a status window in the upper-left corner of the screen. The number in reverse video is unimportant; it's the army number (which may change as the game progresses).

The four numbers underneath are significant, however. The first is the army's active strength (in decimal). The second is the number of injured robots, which will be transported to the botspital of whichever side wins the battle. The third—on the line below—is the number of disrupted robots who will be available for combat in the next turn. The fourth number is how many robots can join the active force two turns from now.

If one side is able to reduce the other player's active force to zero, two things happen. The winner sends all injured bots away to be repaired. The winning side also collects all enemy bots (injured or dazed) and sends them to the reinforcement center to be reprogrammed. Eventually all these bots will available to the winner of this particular battle for future engagements.

Reinforcements And Mergers

At the start of the game, you'll see some armies positioned outside of the hex field. These are reinforcements and reserves in transit to the battle. Player one's reinforcements enter at the bottom right corner; player two's enter at the top left. The line of new armies moves counterclockwise; the army next to the entry point is the next to enter the battlefield.

However, the reinforcements cannot enter the battlefield if an army (friendly or enemy) is blocking their way. Keep your armies off

Lyco Computer Marketing & Consultants

1091 . . . . . $228

SAVE ON THESE IN STOCK PRINTERS

SG-10 . . . . $205

COLOR RIBBONS NOW AVAILABLE!!

PANASONIC
1091	228
3131 (NEW)	264
1092	325
3151	409
1080 (NEW)	209
1592 (NEW)	439

OKIDATA
Okimate 10	179
182	214
192	343
193	563

CITIZEN
MSP-10	255
MSP-15	355
MSP-20	337
MSP-25	495
120D	188
Premier 35	429

EPSON
LX80	209
FX85	333
JX80	Call
Homewriter 10	193
DX10	207
DX20	297
DX35	597
AP-80	244
HI-80	355
HS-80	298
FX-286 (NEW)	489
LQ-800 (NEW)	529
LQ-1000 (NEW)	659

DIABLO
D25	549
630 API	1599
630 ECS	1759
D 80 IF	2395
P 32 CQ1	699
P 38	1749
C 150	999

LEGEND
1080	Call
1380	258
1385	289
808	148

JUKI
Juki 6100	344
RS232 Serial Board	55
6100 Tractor	119
6100 Sheet Feeder	209
Juki 6300	757

TOSHIBA
P1340	469
P351+	1149
P341P	969
P341S	999
351 Sheet Feeder	529

CORONA
LP300	2495
200361 Toner Cartridge	89

SILVER REED
EXP400	249
EXP500	295
EXP550	399
EXP770	749

BROTHER
HR-15XL-P	359
HR-15XL-S	359

C. ITOH
Prowriter 8510 sp+	Call
15505 sp+	Call
Printmaster	Call

STAR MICRONICS
SG-10	205
SG-10c	219
SG-15	367
SD-10	319
SD-15	438
SR 10	469
SR-15	578
SB 10	589
Powertype	297
NX-10 (NEW)	CALL
NB-15 (NEW)	CALL

SEIKOSHA
SP 1000 VC (C 64)	185
SP 1000 A Centronics	199
SP-1000 I IBM	199
SP-1000 AS RS-232	199
SP1000 AP Apple IIc	199
BP-5200 I	649
BP sheet feeder	199
SP 1000 ribbon	8 50
BP-5200 ribbon	12 50

DUST COVERS

Atari
520ST	11 95
130XE	6 99
800XL	6 99
1050	6 99
1025	7 99

Commodore
C128	7 99
1571/1541	6 99
1902	10 95
1702	8 99
C64/Vic20	6 99

Panasonic
1090/1091	8 99
1092	8 99
1093	9 99

Star Micronics
SG/SD10	8 99
SG/SD15	9 99
SR10	9 99
SR15	9 99

Okidata
82/92	8 99
83/93	9 99
193	9 99

DRIVES

INDUS
GT Atari	195
GT Commodore	195

COMMODORE
1571	CALL
1541	CALL

COMTEL
Enhancer 2000 (C 64)	159

TANDON
320K 5¼ Drive	115

INTERFACING

DIGITAL DEVICES
Ape Face XLP (Atari)	49
U-Print A (Atari)	54
U-Print A 16K Buffer	74
U-Print AP 16K (Apple)	99

CARDCO
G-WIZ (C-64)	54
C/?PS (C-64)	49
C/?B (C-64)	39

XETEC
Super Graphix 64	64
Super Graphix JR 64	45

MICROBITS
MPP 1150 (Atari)	54
MPP 1150XL (Atari)	59
MicroPrint (Atari)	39

TYMAC
Connection (C 64)	55
Tackler (Apple)	49
PPC 100 (Apple)	39

MICROTEK
Dumpling GX (Apple)	59
Dumpling 16K (Apple)	89
RV 611C (Apple)	49

ORANGE MICRO
GRAPPLER + (Apple)	85
Grappler 16K (Apple)	149
ORANGE (Apple)	59
Grappler CD (C 64)	79

DISKETTES

DENNISON
ELEPHANT 5¼ SSSD	11 99
ELEPHANT 5¼ SSDD	12 99
ELEPHANT 5¼ DSDD	14 99
PREMIUM 5¼ SSDD	13 99
PREMIUM 5¼ DSDD	15 99

SUNKYONG
SKC 5¼ SSDD	11 99
SKC 5¼ DSDD	13 99

MAXELL
5¼" MD1	13 99

VERBATIM
5¼" SSDD	13 99
5¼ DSDD	19 99

BONUS
5¼" SSDD	8 99
5¼" DSDD	12 99

3.5" DISKETTES

DENNISON
SSDD	5 pak	14 95
SSDD	10 pak	26 95

MAXELL
SSDD	10 pak	29 95
DSDD	10 pak	36 95

3M
SSDD	10 pak	26 95
DSDD	10 pak	32 95

IBM-PC

MICROPROSE (IBM)
F 15 Strike Eagle	20 75
Solo Flight	20 75
Silent Service	20 75
Decision in Desert	24 95
Crusade Europe	24 95

SSI (IBM)
Battle for Normandy	24 95
Knights of Desert	24 95
Tigers in Snow	24 95
Computer Baseball	24 95
Cartels & Cutthroats	24 95
OP Market Garden	29 95
50 Mission Crush	24 95

SUBLOGIC (IBM)
Jet Simulator	34 95
Scenery Disks EA	14 95
Set 1-6	69 95

ACTIVISION (IBM)
Borrowed Time	24 75
Mindshadows	24 75
Music Studio	29 95
Alter Ego	29 95

SYNAPSE (IBM)
Synstock	64 96
Essex	26 95
Wizard of Wall St	28 95
Brimstone	28 95

MICROLEAGUE (IBM)
M L Baseball	24 95
General Mgr	24 95
85 Team Disk	14 95

LEADING EDGE
Nutshell	69 95
Nutshell Filer	149 00

BRODERBUND (IBM)
Bank St Writer	48 95
The Print Shop	34 95
Graphics Library 1	22 95
Ancient Art of War	27 95
Champ Lode Runner	22 95
Karateka	22 95

TOLL FREE 1-800-233-8760

MasterCard

TO ORDER

CALL TOLL FREE 1-800-233-8760
In PA 717-494-1030
Customer Service 717-494-1670

VISA

or send order to
Lyco Computer
P.O. Box 5088
Jersey Shore, PA
17740

RISK FREE POLICY

In-stock items shipped within 24 hours of order No deposit on C O D orders Free shipping on prepaid cash orders within the continental U S Volume discounts available PA residents add sales tax APO FPO and international orders add $5 00 plus 3% for priority mail service Advertised prices show 4% discount for cash add 4% for MasterCard or Visa Personal checks require 4 weeks clearance before shipping Ask about UPS Blue and Red label shipping All merchandise carried under manufacturer s warranty Free catalog with order All items subject to change without notice

your own reinforcement point, and try to block your opponent's armies from this area if you can. If the entry hex is owned but not occupied by your opponent, you'll lose some reinforcements.

After completing a turn, you are credited with additional reinforcements according to how much territory you own. Passing over a hex allows you to claim it; the hex changes color to indicate ownership. Each piece of property provides enough ore and energy to build a new robot, available for use two turns in the future. The numbers in the line of reinforcements are updated after you move to show additional robots being built.

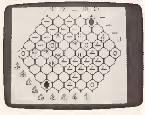
Winning a battle also provides additional armies in the line of reinforcements. As mentioned above, a victorious army captures any dazed enemy bots, which are reprogrammed and available in three turns. At the same time, the winner evacuates injured bots of both sides. Transportation and repair take five turns for friendly bots, seven for enemy bots. The two additional turns are needed for reprogramming the opponent's forces.

If you're losing a battle, the number of injured robots (displayed in the status window) will begin to rise. Remember that, if your opponent reduces your active strength to zero, he or she will capture all of your injured bots; they'll be reprogrammed and added to future reinforcements. To prevent this from happening, you're allowed to bring in a second army for merging. Simply move another army on top of the army with which you want to merge. There's just one rule: One or both of the armies must have a strength less than 32 decimal (1F or less in hex).

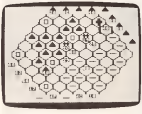
Customizing The Scenarios

The five built-in scenarios provide plenty of variety, but if you'd like to add more challenges, here are some suggestions. (The following line number references are for the 128 version of Hex War, Program 1; other versions may differ slightly, although the variable names are the same in most versions.)

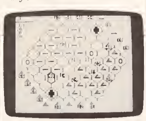
First, a note about the logical organization of the grid. The variables T and B, CT and CB, and HT and HB are used to locate the coordinates on the playing field (see figure). The first number is T (or HT or CT), the second is B (or HB or CB). These coordinates are also used in the three-dimensional MAP array (where level 0 of the array is the army number, 1 is the current owner and 2 keeps track of whether or not a city is located there); they're also part of the ARMY array. By varying the starting position, number of armies, reinforcement strengths, and location of cities, you could simulate historic battles.



The Commodore 128 version of "Hex War," an absorbing strategy game with many variations.



"Hex War" For Atari 600XL, 800, 800XL, 1200XL and 130XE



"Hex War" for the Commodore 64.

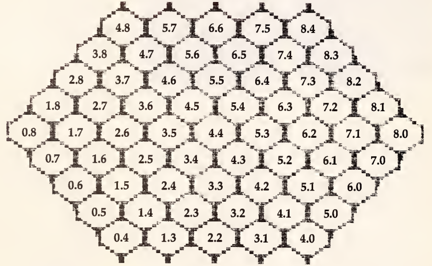
Apple II "Hex War."

To add or subtract cities from the field, change the value of CN in line 50. You'll also have to change the DATA statements in lines 270 and 280. The numbers there are the T and B coordinates of the cities.

The strengths and locations of the armies can be changed as well. The DATA statements starting at line 1540 determine the strength (64) and T/B coordinates for the armies at the beginning of the game. If you wish to start with more armies (or fewer), you'll have to change the inner FOR–NEXT loop (with the index of K) in line 1500. In that same line, change NX(J) to one number higher than the number of armies on each side. For example, if you want six armies apiece, change NX(J) to 7. The subroutine at line 1600 sets up the reinforcements; if you don't like the random patterns, change the formula here.

Variables defined in lines 70–90 control the play of the game. PN determines which player goes second; it can be either zero or one. Variable ME controls the maximum merge strength. If you'd like to be able to merge any two armies, change it to a high value (512, for example). To remove the merge option altogether, change ME to zero.

The movement points are defined by MM in line 80. Movement across friendly territory takes one point, across neutral or hostile territory two points. Increasing MM will give your armies more mobility. The three variables KA, KB, and KC affect the outcome of individual battles. KA determines how many bots are vaporized, KB controls the number injured, and KC affects how many are dazed. If you make the fractions smaller (1/24, for



example), the battles end more quickly. The subroutine starting at line 2600 resolves current battles.

Commodore 64 Version

The 64 version of Hex War (Program 2) looks and plays exactly like the original 128 version. However, one additional step is needed before you run the game. After you have typed in the game and saved it on disk or tape, type this line in direct mode (without line numbers):

POKE 44,64:POKE 64*256,0:NEW

Be sure to press RETURN after you type the line. Now load and run the program as usual. It is very important that you perform this step before running the program: If you don't, the screen will be jumbled and impossible to decipher.

You may find it easier to let the 64 handle this chore for you. Program 3 is a short loader which performs the setup, then loads and runs Hex War. To use the loader, you must have Program 2 saved with the name HEX WAR on the same disk or tape as Program 3 (for tape, Program 2 must follow Program 3 on the tape and the DV=8 in line 10 of Program 3 must be changed to DV=1).

Atari Version

In the Atari version (Program 4), armies are maneuvered using a joystick plugged into port 1. Joystick controls are the same as described above for the 128 version.

This version generates extra colors in graphics mode 0 using a technique known as artifacting. However, the resulting colors may vary on different Ataris, so a small change may be required. The game should start with the red army at the top of the screen and the blue army at the bottom. If that is not the case on your machine, change the following line:

```
ND 4077 RESTORE 4140:FOR A=C
        HSET+240 TO CHSET+24
        7:READ B:POKE A,B:PO
        KE A+8,B*2:NEXT A:RE
        TURN
```

If the colors are not corrected, the machine may appear to declare the wrong side the winner at the end of the game.

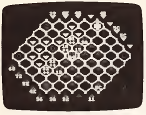
Apple Version

Program 5 is for all Apple II-series computers using either DOS 3.3 or ProDOS. To get the full benefit of the detailed high-resolution graph ics, a color monitor or color TV is recommended. (The program incorporates the HROUT high-resolution graphics routine from the "Apple Superfont" article in the April 1985 issue to generate these graphics.)

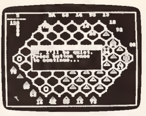
Use keyboard controls to maneuver armies in this version. Press the period (.) key to go northeast (use the > symbol on that key as a reminder), the comma (,) key to go northwest (note the < symbol on that key), the → to move southeast, and the ← key to move southwest. Use the space bar to select or set an army. Press RETURN to end your turn before all your armies have been moved. A player indicator appears in the lower-right corner to indicate which player has the current turn.

IBM PC/PCjr Version

Like the Apple version, the IBM PC/PCjr version of Hex War (Pro-



The IBM PC/PCjr version of "Hex War."



"Hex War" for the 512K Amiga.

gram 6) uses keyboard controls instead of a joystick. Press cursor-up to go northeast, cursor-right to go southeast, cursor-down to go southwest, and cursor-left to go northwest. If you can turn the keyboard 45 degrees in a clockwise direction, these directions will seem more natural.

Use the space bar to select or set an army. Press ESC to end your turn before all your armies have been moved. A player indicator appears in the lower-right corner to indicate which player has the current turn.

Amiga Version

The Amiga version of Hex War (Program 7) requires 512K of memory and is written in Amiga BASIC (not the ABasiC which was shipped with early Amigas). Before you start typing the program, notice the small left arrows at the end of each program line in the listing. These indicate the end of each program line, and are not intended to be typed (in fact, we deliberately chose a character that's not available on the Amiga keyboard). Instead, press RETURN wherever you see a left arrow in the listing.

This game uses exactly the same keyboard controls as the IBM PC/PCjr game (see above). Move the cursor on the playfield with the cursor keys, and press the space bar to pick up or set an army. Your turn ends when all your armies have moved or when you press ESC.

Amiga Hex War includes synthesized speech to emphasize various events and provide information during the game. For instance, when you select an army, the computer tells you the number (in decimal) of robots in that army. If you're not familiar with hexadecimal numbers, this feature can help you learn hex notation.

Either player can turn the speech off or on at any time. Simply click the mouse button twice: A small window appears and the computer announces the current voice status. If speech was previously activated, it now shuts off, and vice versa. Click the mouse button once to erase the window and resume play. A similar window appears to announce the outcome at the end of each game.

For instructions on entering these listings, please refer to "COMPUTE!'s Guide to Typing In Programs" in this issue of COMPUTE!

Program 1. Hex War For Commodore 128

```
MM 10 IF PEEK(46)<>64 THEN GRA
      PHIC1:GRAPHIC0:FAST:FORJ
      =0TO15:COLOR4,J+1:K=J*12
      8:FORL=KTOK+127:BANK14:J
      1=PEEK(53248+L):BANK0:PO
      KE122B8+L,J1:NEXTL,J
FB 20 BANK15:POKE217,4:POKE260
      4,28:SLOW
PO 30 DIMJ,K,HT,HB,CT,CB,J1,J2
      ,A,B,C,D,E
PK 40 OIM ARMY(31,6,1),BTL(64,
      1,3),MAP(9,9,2),FO(20,1)
      ,NX(1),C(2)
BQ 50 CN=12:OIM CIT(CN,1)
BC 60 A=RNO(-TI/97):P0$="{BLU}
      {OFF}[2 P]{DOWN}{2 LEFT}
      [6][*]£":P1$="{YEL}
      {RVS}£[*]{OFF}{DOWN}
      {2 LEFT}{PUR}[2 Y]"
EP 70 PN=1:ME=31
CE 80 MM=3: REM MAX MOVES
JG 90 KA=1/48:KB=1/4B:KC=1/32
BX 100 BANK15:POKE217,4:POKE26
       04,28
SM 110 FORJ=1TO4:REAOA
FQ 120 FORK=ATOA+7:READB:POKEK
       ,B:NEXTK,J
RA 130 OATA 12936,240,240,63,1
       5,3,3,3,3
AS 140 OATA 12984,15,15,252,24
       0,192,192,192,192
PP 150 OATA 12840,3,3,3,3,15,6
       3,240,240
BJ 160 OATA 12944,192,192,192,
       192,240,252,15,15
RG 170 FORJ=0TO63:READK:POKE35
       84+J,K:NEXT
CC 180 OATA0,255,0,15,195,240,
       63,0
KM 190 DATA252,48,0,12,48,0,12
       ,48
AE 200 DATA0,12,48,0,12,48,0,1
       2
GO 210 DATA48,0,12,48,0,12,48,
       0
JA 220 DATA12,48,0,12,48,0,12,
       48
EG 230 DATA0,12,48,0,12,48,0,1
       2
KB 240 DATA48,0,12,48,0,12,63,
       0
8K 250 DATA252,15,195,240,0,25
       5,0,0
PQ 260 FORJ=1TOCN:FORK=0TO1:RE
       AD CIT(J,K):NEXTK:MAP(C
       IT(J,0),CIT(J,1),2)=1:N
       EXTJ: REM T&B OF CITIES
ER 270 OATA 8,4,0,4,8,0,0,8,4,
       0,4,8
JQ 280 OATA 5,5,3,3,6,3,2,5,5,
       2,3,6
XK 300 GOSUB600:GOSU83200:GOSU
       B600:REM GAME
JS 310 CHAR1,6,11:PRINT"{BLU}
       {RVS}{2 SPACES}L[M]
       {2 SPACES}L[Y]
       {2 SPACES}MN{4 SPACES}
       [G][M]{2 SPACES}NM
       {2 SPACES}OM{2 SPACES}"
EP 320 CHAR1,6,12:PRINT"{PUR}
       {RVS}{2 SPACES}[G][M]
       {2 SPACES}L[P]
       {2 SPACES}NM{4 SPACES}N
       M{2 SPACES}O[M]
       {2 SPACES}[G]M
       {2 SPACES}{HOME}{REO}
       {OFF}";
SM 330 PRINTSPC(10);"PLEASE WA
       IT A MOMENT"
FJ 340 GOSUB1500
ME 400 DO
GR 410 POKE208,0
QX 420 GOSUB1900:GOSU8600:GOSU
       B1710:REM FINO BATTLES
QG 430 COLOR4,7-2*PN:GOSUB800:
       REM JOYSTICK
CG 440 FAST:COLOR4,9:GOSUB2100
       :REM BATTLES AGAIN
JJ 450 COLOR4,3:GOSUB2600:REM
       {SPACE}RESOLVE
EB 460 COLOR4,1:GOSUB2100:REM
       {SPACE}POST-BATTLE
AA 470 COLOR4,16:GOSUB2200:REM
        SPLIT PRISONERS
JP 480 GOSUB 2900:REM REINFORC
       EMENTS
HB 490 SLOW:COLOR4,8:GOSUB3400
H?S 500 PN=1-PN
OS 510 LOOP
JA 600 COLOR4,2:COLOR0,2:COLOR
       5,16:SCNCLR:PRINT:PRINT
       :SPRITE1,0
CB 610 FORJ=1TO5:PRINTSPC(13-2
       *J);:FORK=1TOJ+3:PRINT"
       ER{2 SPACES}";:NEXTK:PR
       INT"ER"
CS 620 PRINTSPC(12-2*J);:FORK=
       1TOJ+4:PRINT"W
```

```
         {2 SPACES}Q";:NEXTK:PRI
         NT:NEXTJ
QX 630 FORJ=1TO5:PRINTSPC(J*2)
         ;:FORK=1TO10-J:PRINT"R
         {2 SPACES}E";:NEXTK:PRI
         NT
SS 640 PRINTSPC(J*2+1);:FORK=1
         TO9-J:PRINT"QW
         {2 SPACES}";:NEXTK:PRIN
         T"QW"
PX 650 NEXTJ
CO 660 C$="UI{DOWN}{2 LEFT}JK"
         :O$="{RVS}{V}{C}{DOWN}
         {2 LEFT}{F}{O}"
MJ 670 COLOR5,3:FORJ=1TO12:GOS
         UB710:NEXT
HQ 680 J=1:COLOR5,7:IFGN=1THEN
         C$=O$:GOSUB710:J=2:COLO
         R5,5:GOSUB710
RB 690 IFGN=2THENC$=O$:GOSUB71
         0:J=3:COLOR5,5:GOSUB710
SH 700 PRINT"{HOME}";:RETURN
KB 710 K=CIT(J,0):L=CIT(J,1):X
         =(K-L)*2+19:Y=(12-(K+L)
         )*2+3:CHAR1,X,Y,C$:RETU
         RN
FF 800 IFNX(PN)<2THENRETURN
AX 805 HT=4:HB=4:GOSUB1000:SPR
         ITE1,1,1,0
KE 810 MV=0:CT=0:CB=0:PK=0:REM
          PICKED UP OR NOT
HS 820 K=0:FORJ=1TONX(PN)-1:IF
          (ARMY(J,0,PN)>0)AND(AR
         MY(J,6,PN)<1)THENK=1:J=N
         X(PN)-1
KX 830 NEXTJ:IFK=0 THEN RETURN
SB 840 GETG$:IFG$=CHR$(13)THEN
          RETURN
HD 850 J=JOY(2):IFJ=0THEN840:E
         LSE IF(JAND128)THEN1100
          :ELSE IF(JAND1)=0THEN84
         0
KK 860 J=(J-1)/2:IFJAND1THENB1
         =HB+J-2:T1=HT:ELSET1=HT
         +1-J:B1=HB
HO 870 IF (T1<0)OR(T1>8)THEN84
         0
FP 880 IF (B1<0)OR(B1>8)THEN84
         0
KC 890 S1=T1+B1:IF(S1<4)OR(S1>
         12)THEN840
EM 900 HB=B1:HT=T1:GOSUB1000:W
         INDOW1,1,8,4,1:QN=MAP(H
         T,HB,0):IFQN=0THENPRINT
          "{2 HOME}";:GOTO840:ELS
         E Q1=MAP(HT,HB,1)-1
JF 910 COLOR5,7-2*Q1:PRINTUSIN
         G"{RVS}##";QN;:PRINT"**
         ****{OFF}{5}";
RH 920 FORJ=0TO3:PRINTUSING"##
         ##";ARMY(QN,J,Q1);:NEXT
OO 930 PRINT"{2 HOME}";:GOTO84
         0
PA 1000 SX=172+16*(HT-HB):SY=2
         64-16*(HT+HB)
KC 1010 MOVSPR1,SX,SY
SA 1020 IF MAP(HT,HB,2)=1THENS
         PRITE1,1,3,0:RETURN
KP 1030 SPRITE1,1,1,0:RETURN
CX 1100 IFJOY(2)<>0THEN1100
FE 1110 IFPK=1THEN1200:REM PIC
         KEDUP, CHECK IF OK
FO 1120 IF((MAP(HT,HB,1)<>PN+1
         )OR(MAP(HT,HB,0)=0))TH
         ENGOTO810:REM NO ONE H
         OME
XE 1130 AN=MAP(HT,HB,0):IFARMY
         (AN,6,PN)<>0THEN810:RE
         M ARMY AN IS ENGAGED
SO 1140 PK=1:CT=HT:CB=HB:CS=AR
         MY(AN,0,PN):REM T&B, C
         URRENT STRENGTH
OP 1150 SOUND1,5000,10
MK 1160 GOTO840
XP 1200 J=((HT=CT)AND(HB=CB))
AQ 1210 IFJAND(MV=0)THEN810
RE 1220 IFJAND(MV>0)THEN1420
MH 1230 AS=ARMY(MAP(HT,HB,0),0
         ,PN):IF((AS>ME)AND(CS>
         ME))OR((MAP(HT,HB,1)-1
         =1-PN)AND(AS>0)) THEN8
         40
FB 1240 DT=ABS(CT-HT):DB=ABS(C
         B-HB):TL=DB+DT:IF NOT(
         (TL=1)OR((CT+CB=HT+HB)
         AND(DT=1)))THEN840
FS 1250 MG=MAP(HT,HB,0): IF MG
         =0THEN1300
AE 1260 FORJ=0TO3:ARMY(MG,J,PN
         )=ARMY(MG,J,PN)+ARMY(A
         N,J,PN):ARMY(AN,J,PN)=
         0:NEXTJ
SM 1270 ARMY(MG,6,PN)=1:MAP(CT
         ,CB,0)=0
AD 1280 CS=ARMY(MG,0,PN):AN=MG
         :MV=MM+1
RH 1290 GOTO 1380
FF 1300 NB=MAP(HT,HB,1)-1:MV=M
         V+1:IF(NB<>PN)THENMV=M
         V+1
OK 1310 MAP(CT,CB,0)=0
SA 1320 MAP(HT,HB,0)=AN:MAP(HT
         ,HB,1)=PN+1:ARMY(AN,4,
         PN)=HT:ARMY(AN,5,PN)=H
         B:IF MV>=MMTHEN ARMY(A
         N,6,PN)=1
KR 1330 K=0:FORJ=-1TO1STEP2:J1
         =HT+J:J2=HB+J:J3=HB-J:
         IF(J1<0)OR(J1>8)THEN13
         40:ELSE IF(MAP(J1,HB,0
         )>0)THENIF(MAP(J1,HB,1
         )=2-PN)THENK=1:J=1:GOT
         O1360
BR 1340 IF(J2<0)OR(J2>8)THEN13
         50:ELSE IF(MAP(HT,J2,0
         )>0)THENIF(MAP(HT,J2,1
         )=2-PN)THENK=1:J=1:GOT
         O1360
KE 1350 IF(J3<0)OR(J3>8)OR(J1<
         0)OR(J1>8)THEN1360:ELS
         E IF(MAP(J1,J3,0)>0)TH
         ENIF(MAP(J1,J3,1)=2-PN
         )THENK=1:J=1
HK 1360 NEXTJ: REM ZOC
AS 1370 IFK=1THEN ARMY(AN,6,PN
         )=1:MV=MM+1
EG 1380 A=PN:J=CT:K=CB:C=0:D=0
         :GOSUB1830
QP 1390 J=HT:K=HB:C=CS:D=ARMY(
         AN,6,PN):GOSUB1830
XJ 1400 CT=HT:CB=HB
MQ 1410 IFMV<MMTHEN840
HA 1420 ARMY(AN,6,PN)=1:J=HT:K
         =HB:C=CS:D=1:GOSUB1830
SG 1430 GOTO810
FC 1500 RESTORE1540:FORJ=0TO1:
         NX(J)=5:FORK=1TO4:READ
         A,B,C
QM 1510 ARMY(K,0,J)=A:ARMY(K,4
         ,J)=B:ARMY(K,5,J)=C:MA
         P(B,C,0)=K:MAP(B,C,1)=
         J+1
BJ 1520 NEXTK,J
EK 1530 REM STRENGTH, T-POS, B
         -POS
JO 1540 DATA 64,2,8,64,3,7,64,
         5,6,64,6,6:REM BLUE
QP 1550 DATA 64,2,2,64,3,2,64,
         5,1,64,6,0:REM VIOLET
JK 1600 REM SET RANDOM REINFOR
         CEMENTS
AS 1610 FORJ=0TO1:FORK=0TO20
QM 1620 A=INT(RND(1)*K*3):FORL
         =1TO5:A=A+INT(RND(1)*2
         1-8):NEXTL:IFA<16THENA
         =0:ELSEA=(A+K*8)AND254
HA 1630 FQ(K,J)=A:NEXTK,J
RG 1640 RETURN
EH 1700 REM ARMIES->MAP
SC 1710 FORJ=0TO8:FORK=0TO8
FX 1720 A=MAP(J,K,1):IFATHENA=
         A-1:GOSUB1800
HH 1730 NEXTK,J
PH 1740 FORA=0TO1:E=13+A*12:F=
         A*22:OX=2-4*A:O=0
PF 1750 FORJ=0TO8:C=FQ(J,A):GO
         SUB1840
HF 1760 E=E+OX*2:IFJ>3THENF=F+
         OX:E=E-OX
XC 1770 NEXTJ,A
CA 1780 RETURN
QC 1800 B=MAP(J,K,0)
PH 1810 C=ARMY(B,0,A)
BK 1820 D=ARMY(B,6,A)
FJ 1830 E=(J-K+10)*2-1:F=(13-J
         -K)*2+1: REM T&B TO X/
         Y
SC 1840 CHAR1,E,F:IFATHENPRINT
         P1$:ELSEPRINTP0$
MH 1850 IFC=0THENRETURN
PP 1860 COLOR5,(7-2*A):CHAR1,E
         ,F+A
HR 1870 PRINTCHR$(18);RIGHT$(H
         EX$(C),2);CHR$(146)
RD 1880 IFDTHENF=F+1-A:G=1024+
         E+F*40:POKEG,43:POKEG+
         1,43:REM ++
XK 1890 RETURN
GC 1900 SW=0:E=NX(PN)-1:IFE<1T
         HENRETURN
XJ 1910 FORJ=1TOE-1:IFARMY(J,0
         ,PN)<1THENBEGIN
SE 1920 T=ARMY(J,4,PN):B=ARMY(
         J,5,PN):IFMAP(T,B,0)=J
         THENMAP(T,B,0)=0
BP 1930 FORK=JTOE:FORL=0TO6:AR
         MY(K,L,PN)=ARMY(K+1,L,
         PN):ARMY(K+1,L,PN)=0:N
         EXTL
JQ 1940 T=ARMY(K,4,PN):B=ARMY(
         K,5,PN):MAP(T,B,0)=K
PH 1950 NEXTK
MA 1960 NX(PN)=NX(PN)-1:J=E:SW
         =1:BEND
FJ 1970 NEXTJ:IF SW THEN1900
MG 2000 FORJ=1TOE:ARMY(J,0,PN)
         =ARMY(J,0,PN)+ARMY(J,2
         ,PN)
SO 2010 ARMY(J,2,PN)=ARMY(J,3,
         PN):ARMY(J,3,PN)=0
QS 2020 ARMY(J,6,PN)=0
PP 2030 NEXTJ:K=NX(1-PN):FOR J
         =1TOK:ARMY(J,6,1-PN)=0
         :NEXT
XF 2040 GOSUB2400
DS 2050 IFBP>0 THEN FORJ=0TO1:
         FORK=1TOBP:A=BTL(K,J,0
         ):ARMY(A,6,J)=ARMY(A,6
         ,J)+1:NEXTK,J
CA 2060 RETURN
KG 2100 GOSUB2400
BX 2110 A=NX(0):IFNX(1)>ATHENA
         =NX(1)
EX 2120 FORJ=0TO1:FORK=1TOA:AR
         MY(K,6,J)=0:NEXTK,J
GP 2130 GOSUB2050
SH 2140 RETURN
PB 2200 FORJ=0TO1:A=1-J:B=NX(J
         )-1
OO 2210 FORK=1TOB
EM 2220 IF ARMY(K,0,J)<1 THEN
         {SPACE}BEGIN
JC 2230 FQ(2,A)=FQ(2,A)+ARMY(K
```

```
            ,2,J)+ARMY(K,3,J):IF F
            Q(2,A)>255 THEN C=FQ(2
            ,A)-255:FQ(3,A)=FQ(3,A
            )+C:FQ(2,A)=255
JX 2240 FQ(6,A)=FQ(6,A)+ARMY(K
            ,1,J):IF FQ(6,A)>255 T
            HEN C=FQ(6,A)-255:FQ(7
            ,A)=FQ(7,A)+C:FQ(6,A)=
            255
JQ 2250 IF (MAP(ARMY(K,4,J),AR
            MY(K,5,J),Ø)=K)AND(MAP
            (ARMY(K,4,J),ARMY(K,5,
            J),1)=J+1) THEN MAP(AR
            MY(K,4,J),ARMY(K,5,J),
            Ø)=Ø
RB 2260 FORL=ØTO6:ARMY(K,L,J)=
            Ø:NEXTL
BB 2270 BEND
FR 22BØ IF ARMY(K,6,J)<1 THEN
            {SPACE}BEGIN:REM EVACU
            ATE INJURED
AE 229Ø FQ(4,J)=FQ(4,J)+ARMY(K
            ,1,J): ARMY(K,1,J)=Ø
DJ 23ØØ IF FQ(4,J)>255 THEN C=
            FQ(4,J)-255:FQ(5,J)=FO
            (5,J)+C:FQ(4,J)=255
KA 231Ø BEND
XP 232Ø NEXTK,J:RETURN
SJ 24ØØ BP=Ø
KG 241Ø FORJ=ØTOB:J1=(J-4)*(4-
            J>Ø):J2=8-(J>4)*(4-J):
            FORK=J1TOJ2
JE 242Ø A=MAP(J,K,Ø)
HQ 243Ø R=MAP(J,K,1)
BG 2440 IF (A=Ø)OR(R=Ø) THEN24
            9Ø
KQ 245Ø IF ARMY(A,Ø,R-1)<1 THE
            N249Ø
DK 246Ø T=J+1:B=K:GOSUB25ØØ
HK 247Ø B=B-1:GOSUB25ØØ
BP 248Ø T=T-1:GOSUB25ØØ
RH 249Ø NEXTK,J:RETURN
RR 25ØØ IF(T<Ø)OR(B<Ø)OR(T>8)O
            R(B>8)THEN RETURN
MA 251Ø PA=MAP(T,B,Ø):IF PA=Ø
            {SPACE}THEN RETURN
FG 252Ø IF MAP(T,B,1)=R THEN R
            ETURN
FX 253Ø IF ARMY(PA,Ø,2-R)<1 TH
            EN RETURN
MA 254Ø BP=BP+1:BTL(BP,R-1,Ø)=
            A:BTL(BP,2-R,Ø)=PA:RET
            URN
CF 26ØØ IFBP=ØTHENRETURN
FC 261Ø FORJ=1TOBP
XM 262Ø FORK=ØTO1:A=1-K
QX 263?Ø AN=BTL(J,K,Ø)
BF 264Ø AS=ARMY(AN,Ø,K):HT=ARM
            Y(AN,6,K):CT=INT(AS/HT
            )+1
GH 265Ø BTL(J,A,1)=INT(CT*KA+1
            )
JC 266Ø BTL(J,A,2)=INT(CT*KB+1
            )
MB 267Ø BTL(J,A,3)=INT(CT*KC+1
            )
XC 268Ø NEXTK,J
MD 27ØØ FORJ=1TOBP:JØ=BTL(J,Ø,
            Ø):J1=BTL(J,1,Ø)
BR 271Ø GOSUB31ØØ
CG 272Ø ARMY(JØ,Ø,Ø)=ARMY(JØ,Ø
            ,Ø)-A*BTL(J,Ø,1)
SK 273Ø ARMY(J1,Ø,1)=ARMY(J1,Ø
            ,1)-B*BTL(J,1,1)
MS 274Ø GOSUB31ØØ
BC 275Ø C=A*BTL(J,Ø,2):ARMY(JØ
            ,Ø,Ø)=ARMY(JØ,Ø,Ø)-C:A
            RMY(JØ,1,Ø)=ARMY(JØ,1,
            Ø)+C
FJ 276Ø C=B*BTL(J,1,2):ARMY(J1
            ,Ø,1)=ARMY(J1,Ø,1)-C:A
            RMY(J1,1,1)=ARMY(J1,1,
            1)+C
QD 277Ø GOSUB31ØØ
AX 278Ø C=A*BTL(J,Ø,3):ARMY(JØ
            ,Ø,Ø)=ARMY(JØ,Ø,Ø)-C:A
            RMY(JØ,3,Ø)=ARMY(JØ,3,
            Ø)+C
FF 279Ø C=B*BTL(J,1,3):ARMY(J1
            ,Ø,1)=ARMY(J1,Ø,1)-C:A
            RMY(J1,3,1)=ARMY(J1,3,
            1)+C
XP 28ØØ NEXTJ
CB 281Ø RETURN
GQ 29ØØ A=1-PN:B=Ø
GG 291Ø FORJ=ØTO8:FORK=ØTO8
PA 292Ø IFMAP(J,K,1)=PN+1THENB
            =B+1
KD 293Ø NEXTK,J
JB 295Ø FQ(1,PN)=FQ(1,PN)+B: I
            F FQ(1,PN)>255 THEN B=
            FQ(1,PN)-255:FQ(2,PN)=
            FQ(2,PN)+B:FQ(1,PN)=25
            5
GQ 296Ø T=4:B=PN*8
SJ 297Ø IF MAP(T,B,Ø)<>Ø THEN
            {SPACE}RETURN
JK 298Ø IF MAP(T,B,1)=PN+1 THE
            N FQ(Ø,A)=Ø:FQ(1,A)=Ø:
            GOTO3Ø6Ø
XS 299Ø J=NX(A):IFJ>31THENRETU
            RN
HR 3ØØØ J1=FQ(Ø,A):IF J1<1 THE
            N3Ø6Ø
EP 3Ø1Ø NX(A)=NX(A)+1
XE 3Ø2Ø MAP(T,B,Ø)=J:MAP(T,B,1
            )=A+1
BK 3Ø3Ø ARMY(J,Ø,A)=J1
QR 3Ø4Ø FORK=1TO3:ARMY(J,K,A)=
            Ø:NEXTK
PM 3Ø5Ø ARMY(J,4,A)=T:ARMY(J,5
            ,A)=B
KH 3Ø6Ø FORK=ØTO19:FQ(K,A)=FQ(
            K+1,A):NEXTK
PK 3Ø7Ø FQ(2Ø,A)=Ø
BA 3Ø8Ø RETURN
FS 31ØØ A=Ø:FORM=1TO6:IFRND(1)
            <.5THENA=A+1
AH 311Ø NEXTM:B=6-A:RETURN
PS 32ØØ WINDOW6,7,33,16,1
EJ 321Ø PRINT"Ɛ7ƎƐ2 MƎƐ2 LƎ
            {RVS}Ɛ2 KƎƐJƎƐHƎƐGƎ SC
            ENARIO ƐMƎƐNƎƐLƎ{OFF}
            Ɛ3 KƎƐJƎƐ2 GƎƐMƎ"
FB 322Ø PRINT"{LEFT}ƐGƎƐMƎ
            {2 SPACES}1> CAPTURE C
            APITAL/FAR{2 SPACES}
            ƐGƎ";
QX 323Ø PRINT"ƐMƎ{2 SPACES}2>
            {SPACE}CAPTURE CAPITAL
            /NEAR ƐGƎ";
GQ 324Ø PRINT"ƐMƎ{2 SPACES}3>
            {SPACE}OCCUPY
            {3 SPACES}8/12 CITIES
            {SPACE}ƐGƎ";
AR 325Ø PRINT"ƐMƎ{2 SPACES}4>
            {SPACE}CONTROL
            {2 SPACES}6/12 CITIES
            {SPACE}ƐGƎ";
DJ 326Ø PRINT"ƐMƎ{2 SPACES}5>
            {SPACE}OCCUPY
            {2 SPACES}4Ø/61 HEXES
            {2 SPACES}ƐGƎƐMƎ";:PRI
            NTSPC(26);"ƐGƎ"
SQ 327Ø PRINT"Ɛ2 MƎƐ2 LƎ{RVS}
            Ɛ2 KƎƐJƎƐHƎƐGƎ
            {1Ø SPACES}ƐMƎƐNƎƐLƎ
            {OFF}Ɛ3 KƎƐJƎƐ2 GƎ"
GE 32BØ GETKEYA$:GN=VAL(A$):IF
            GN<1ORGN>5THEN328Ø
DP 329Ø CHAR1,2,1+GN,"ZZ":SLEE
            P1:PRINT"{2 HOME}"
PX 33ØØ RETURN
GS 34ØØ A=Ø:ON GN GOSUB 343Ø,3
            45Ø,348Ø,349Ø,358Ø
JD 341Ø IFA=ØTHENRETURN:ELSEEN
            $=C$:QQ=A:GOSUB6ØØ:GOS
            UB171Ø:A=QQ
KD 342Ø PRINT"{HOME}PLAYER";A;
            " WINS":PRINTEN$:PRINT
            "(PRESS ANY KEY)":POKE
            2Ø8,Ø:GETKEYA$:RUN
BB 343Ø IF MAP(CIT(2,Ø),CIT(2,
            1),1)=1THENA=2:C$="BLU
            E CAPTURED THE CAPITAL
            ":RETURN
CR 344Ø GOTO346Ø
SG 345Ø IF MAP(CIT(3,Ø),CIT(3,
            1),1)=1THENA=2:C$="BLU
            E CAPTURED THE CAPITAL
            ":RETURN
KB 346Ø IF MAP(CIT(1,Ø),CIT(1,
            1),1)=2THENA=1:C$="VIO
            LET CAPTURED THE CAPIT
            AL"
FP 347Ø RETURN
BC 348Ø L=8:GOTO35ØØ
JB 349Ø L=6
FG 35ØØ C(1)=Ø:C(2)=Ø
PK 351Ø FORJ=1TO12:T=CIT(J,Ø):
            B=CIT(J,1)
MB 352Ø R=MAP(T,B,1):C(R)=C(R)
            +1
DB 353Ø IF GN=4THEN AN=MAP(T,B
            ,Ø):IF R>Ø THEN IF (AN
            =Ø)OR(ARMY(AN,6,R-1)>Ø
            )THEN C(R)=C(R)-1
KJ 354Ø NEXTJ
CB 355Ø IF C(1)=> L THEN A=2:C
            $="BLUE HAS CAPTURED"+
            STR$(C(1))+" CITIES":R
            ETURN
QF 356Ø IF C(2)=> L THEN A=1:C
            $="VIOLET HAS CAPTURED
            "+STR$(C(2))+" CITIES"
AB 357Ø RETURN
RK 358Ø C(1)=Ø:C(2)=Ø
PP 359Ø FORJ=ØTO8:FORK=ØTO8
KM 36ØØ R=MAP(J,K,1):C(R)=C(R)
            +1
SR 361Ø NEXTK,J
CE 362Ø IF C(1)=>4ØTHENA=2:C$=
            "BLUE OCCUPIES"+STR$(C
            (1))+" HEXES":RETURN
RK 363Ø IF C(2)=>4ØTHENA=1:C$=
            "VIOLET OCCUPIES"+STR$
            (C(2))+" HEXES"
DF 364Ø RETURN
```

Program 2. Hex War For Commodore 64

Version by Kevin Mykytyn, Editorial Programmer

```
EB Ø POKE53269,Ø:PRINT"{CLR}":
         XC=14:YC=12:GOSUB1:PRINT"
         PLEASE WAIT":GOTO 9
ME 1 POKE781,YC:POKE782,XC:POK
         E783,Ø:SYS6552Ø:RETURN
GB 2 QV=15-(PEEK(5632Ø)AND15):
         J=JY(QV)-128*((PEEK(5632Ø
         )AND16)=Ø):RETURN
JQ 3 OV=OVAND255:H$="Ø12345678
         9ABCDEF":Z$=MID$(H$,INT(Q
         V/16)+1,1)
EH 4 Z$=Z$+MID$(H$,OV-INT(QV/1
         6)*16+1,1):RETURN
XC 9 IFPEEK(122B9)+PEEK(1229Ø)
         =212THEN2Ø
```

```
GJ 10 POKE56334,0:POKE1,51:FOR
      I=0TO2047:POKEI+12288,PE
      EK(I+53248):NEXT:POKE1,5
      5
KS 20 POKE56334,1:POKE53272,28
      :C4=53280:C0=53281:C5=64
      6:SC=53287:POKE2040,13
AC 25 FORA=0TO10:READJY(A):NEX
      T:DATA 0,1,5,0,7,8,6,0,3
      ,2,4
KM 26 FORA=54272TO54295:POKEA,
      0:NEXT:POKEA,15:POKE5427
      3,40:POKE54277,25
PD 30 DIMJ,K,HT,HB,CT,CB,J1,J2
      ,A,B,C,D,E
PK 40 DIM ARMY(31,6,1),BTL(64,
      1,3),MAP(9,9,2),FQ(20,1)
      ,NX(1),C(2)
BQ 50 CN=12:DIM CIT(CN,1)
BC 60 A=RND(-TI/97):P0$="{BLU}
      {OFF}ɛ2 Pɜ{DOWN}{2 LEFT}
      ɛ6ɜɛ*ɜ£":P1$="{YEL}
      {RVS}£ɛ*ɜ{OFF}{DOWN}
      {2 LEFT}{PUR}ɛ2 Yɜ"
EP 70 PN=1:ME=31
CE 80 MM=3: REM MAX MOVES
JG 90 KA=1/48:KB=1/48:KC=1/32
SM 110 FORJ=1TO4:READA
FQ 120 FORK=ATOA+7:READB:POKEK
       ,B:NEXTK,J
RA 130 DATA 12936,240,240,63,1
       5,3,3,3,3
AS 140 DATA 12984,15,15,252,24
       0,192,192,192,192
PP 150 DATA 12840,3,3,3,3,15,6
       3,240,240
BJ 160 DATA 12944,192,192,192,
       192,240,252,15,15
DQ 170 FORJ=0TO63:READK:POKE83
       2+J,K:NEXT
CC 180 DATA0,255,0,15,195,240,
       63,0
KM 190 DATA252,48,0,12,48,0,12
       ,48
AE 200 DATA0,12,48,0,12,48,0,1
       2
GD 210 DATA48,0,12,48,0,12,48,
       0
JA 220 DATA12,48,0,12,48,0,12,
       48
EG 230 DATA0,12,48,0,12,48,0,1
       2
KB 240 DATA48,0,12,48,0,12,63,
       0
BK 250 DATA252,15,195,240,0,25
       5,0,0
KF 260 FORJ=1TOCN:FORK=0TO1:RE
       AD CIT(J,K):NEXTK:MAP(C
       IT(J,0),CIT(J,1),2)=1:N
       EXTJ
ER 270 DATA 8,4,0,4,8,0,0,8,4,
       0,4,8
JQ 280 DATA 5,5,3,3,6,3,2,5,5,
       2,3,6
FP 300 GOSUB600:GOSUB3200:GOSU
       B600
HD 310 XC=6:YC=11:GOSUB1:PRINT
       "{BLU}{RVS}{2 SPACES}L
       ɛMɜ{2 SPACES}LɛYɜ
       {2 SPACES}MN{4 SPACES}
       ɛGɜɛMɜ{2 SPACES}NM
       {2 SPACES}OM{2 SPACES}"
CH 320 XC=6:YC=12:GOSUB1:PRINT
       "{PUR}{RVS}{2 SPACES}
       ɛGɜɛMɜ{2 SPACES}LɛPɜ
       {2 SPACES}NM{4 SPACES}N
       M{2 SPACES}OɛMɜ
       {2 SPACES}ɛGɜM
       {2 SPACES}{HOME}{RED}
       {OFF}";
SM 330 PRINTSPC(10);"PLEASE WA
       IT A MOMENT"
FJ 340 GOSUB1500
SB 410 POKE198,0
QX 420 GOSUB1900:GOSUB600:GOSU
       B1710:REM FIND BATTLES
BR 430 POKEC4,6-2*PN:GOSUB800:
       REM JOYSTICK
PF 440 POKEC4,8:GOSUB2100:REM
       {SPACE}BATTLES AGAIN
PB 450 POKEC4,3:GOSUB2600:REM
       {SPACE}RESOLVE
BP 460 POKEC4,0:GOSUB2100:REM
       {SPACE}POST-BATTLE
EM 470 POKEC4,15:GOSUB2200:REM
        SPLIT PRISONERS
JP 480 GOSUB 2900:REM REINFORC
       EMENTS
SR 490 POKEC4,7:GOSUB3400
HS 500 PN=1-PN
KC 510 GOTO410
AC 600 POKEC4,1:POKEC0,1:POKEC
       5,15:PRINT"{CLR}";:PRIN
       T:PRINT:POKESC,0
CB 610 FORJ=1TO5:PRINTSPC(13-2
       *J);:FORK=1TOJ+3:PRINT"
       ER{2 SPACES}";:NEXTK:PR
       INT"ER"
CS 620 PRINTSPC(12-2*J);:FORK=
       1TOJ+4:PRINT"W
       {2 SPACES}Q";:NEXTK:PRI
       NT:NEXTJ
QX 630 FORJ=1TO5:PRINTSPC(J*2)
       ;:FORK=1TO10-J:PRINT"R
       {2 SPACES}E";:NEXTK:PRI
       NT
SS 640 PRINTSPC(J*2+1);:FORK=1
       TO9-J:PRINT"QW
       {2 SPACES}";:NEXTK:PRIN
       T"QW"
PX 650 NEXTJ
CD 660 C$="UI{DOWN}{2 LEFT}JK"
       :D$="{RVS}ɛVɜɛCɜ{DOWN}
       {2 LEFT}ɛFɜɛDɜ"
QC 670 POKEC5,2:FORJ=1TO12:GOS
       UB710:NEXT
FJ 680 J=1:POKEC5,6:IFGN=1THEN
       C$=D$:GOSUB710:J=2:POKE
       C5,4:GOSUB710
RO 690 IFGN=2THENC$=D$:GOSUB71
       0:J=3:POKEC5,4:GOSUB710
SH 700 PRINT"{HOME}";:RETURN
PD 710 K=CIT(J,0):L=CIT(J,1):X
       =(K-L)*2+19:Y=(12-(K+L)
       )*2+3
RB 715 XC=X:YC=Y:GOSUB1:PRINTC
       $:RETURN
FF 800 IF NX(PN)<2 THEN RETURN
CQ 805 HT=4:HB=4:GOSUB1000:POK
       E53269,1:POKESC,0
DJ 810 MV=0:CT=0:CB=0:PK=0:K=0
AX 820 FORJ=1TONX(PN)-1:IF(ARM
       Y(J,0,PN)>0)AND(ARMY(J,
       6,PN)<1)THENK=1:J=NX(PN
       )-1
KX 830 NEXTJ:IFK=0 THEN RETURN
SB 840 GETG$:IFG$=CHR$(13)THEN
        RETURN
MA 850 GOSUB2:IFJ=0THEN840
FD 855 IF(JAND128)THEN1100
BR 857 IF(JAND1)=0THEN840
DD 860 J=(J-1)/2:IFJAND1THENB1
       =HB+J-2:T1=HT:GOTO870
AE 865 T1=HT+1-J:B1=HB
HD 870 IF (T1<0)OR(T1>8)THEN84
       0
FP 880 IF (B1<0)OR(B1>8)THEN84
       0
KC 890 S1=T1+B1:IF(S1<4)OR(S1>
       12)THEN840
QR 895 HB=B1:HT=T1:GOSUB1000
SC 896 PRINT "{HOME}";:FORZ=1T
       O6:PRINT"{8 SPACES}":NE
       XT
QE 900 QN=MAP(HT,HB,0):IFQN=0T
       HEN840
KD 905 Q1=MAP(HT,BB,1)-1
FF 910 POKEC5,6-2*Q1:PRINT"
       {HOME}"QN:PRINT"******
       {OFF}ɛ5ɜ"
GM 920 FORJ=0TO3:PRINTARMY(QN,
       J,Q1):NEXT
QB 930 GOTO840
PA 1000 SX=172+16*(HT-HB):SY=2
        64-16*(HT+HB)
BK 1010 POKE53249,0:POKE53248,
        SXAND255:POKE53264,-(S
        X>255):POKE53249,SY
FF 1020 IF MAP(HT,HB,2)=1THENP
        OKESC,2:RETURN
HG 1030 POKE53269,1:POKESC,0:R
        ETURN
RR 1100 GOSUB2:IFJ<>0THEN1100
KB 1110 IFPK=1THEN1200
FK 1120 IF((MAP(HT,HB,1)<>PN+1
        )OR(MAP(HT,HB,0)=0))TH
        ENGOTO810
SE 1130 AN=MAP(HT,HB,0):IFARMY
        (AN,6,PN)<>0THEN810
SJ 1140 PK=1:CT=HT:CB=HB:CS=AR
        MY(AN,0,PN)
EX 1150 POKE54276,32:POKE54276
        ,33
MK 1160 GOTO840
XP 1200 J=((HT=CT)AND(HB=CB))
AQ 1210 IFJAND(MV=0)THEN810
RE 1220 IFJAND(MV>0)THEN1420
DG 1225 AS=ARMY(MAP(HT,HB,0),0
        ,PN)
CA 1230 IF((AS>ME)AND(CS>ME))O
        R((MAP(HT,HB,1)-1=1-PN
        )AND(AS>0))THEN840
SE 1235 DT=ABS(CT-HT):DB=ABS(C
        B-HB)
MF 1240 TL=DB+DT:IF NOT((TL=1)
        OR((CT+CB=HT+HB)AND(DT
        =1)))THEN840
FS 1250 MG=MAP(HT,HB,0):IF MG=
        0THEN1300
AE 1260 FORJ=0TO3:ARMY(MG,J,PN
        )=ARMY(MG,J,PN)+ARMY(A
        N,J,PN):ARMY(AN,J,PN)=
        0:NEXTJ
SM 1270 ARMY(MG,6,PN)=1:MAP(CT
        ,CB,0)=0
AD 1280 CS=ARMY(MG,0,PN):AN=MG
        :MV=MM+1
RH 1290 GOTO 1380
FF 1300 N8=MAP(HT,HB,1)-1:MV=M
        V+1:IF(N8<>PN)THENMV=M
        V+1
DK 1310 MAP(CT,CB,0)=0
MF 1320 MAP(HT,HB,0)=AN:MAP(HT
        ,HB,1)=PN+1:ARMY(AN,4,
        PN)=HT:ARMY(AN,5,PN)=H
        B
HE 1322 IFMV>=MMTHENARMY(AN,6,
        PN)=1
RB 1325 K=0:FORJ=-1TO1STEP2:J1
        =HT+J:J2=HB+J:J3=HB-J:
        IF(J1<0)OR(J1>8)THEN13
        40
AR 1327 IF(MAP(J1,HB,0)<=0)THE
        N1340
GK 1330 IF(MAP(J1,HB,1)=2-PN)T
        HENK=1:J=1:GOTO1360
HQ 1340 IF(J2<0)OR(J2>8)THEN13
        50
BH 1342 IF(MAP(HT,J2,0)>0)THEN
        IF(MAP(HT,J2,1)=2-PN)T
        HENK=1:J=1:GOTO1360
SH 1350 IF(J3<0)OR(J3>B)OR(J1<
        0)OR(J1>8)THEN1360
```

```
RP 1352 IF(MAP(J1,J3,Ø)>Ø)THEN
        IF(MAP(J1,J3,1)=2-PN)T
        HENK=1:J=1
RB 1360 NEXTJ
AS 1370 IFK=1THEN ARMY(AN,6,PN
        )=1:MV=MM+1
EG 1380 A=PN:J=CT:K=C8:C=Ø:D=Ø
        :GOSU81830
QP 1390 J=HT:K=HB:C=CS:D=ARMY(
        AN,6,PN):GOSUB1830
XJ 1400 CT=HT:C8=HB
MQ 1410 IFMV<MMTHEN840
HA 1420 ARMY(AN,6,PN)=1:J=HT:K
        =HB:C=CS:D=1:GOSUB1830
SG 17430 GOTO810
GR 1500 RESTORE:FORZ=1TO135:RE
        AD8:NEXT:FORJ=ØTO1:NX(
        J)=5:FORK=1TO4:READA,8
        ,C
QM 1510 ARMY(K,Ø,J)=A:ARMY(K,4
        ,J)=8:ARMY(K,5,J)=C:MA
        P(8,C,Ø)=K:MAP(8,C,1)=
        J+1
87J 1520 NEXTK,J
EK 1530 REM STRENGTH, T-POS, 8
        -POS
JD 1540 DATA 64,2,8,64,3,7,64,
        5,6,64,6,6:REM BLUE
QP 1550 DATA 64,2,2,64,3,2,64,
        5,1,64,6,Ø:REM VIOLET
JK 1600 REM SET RANDOM REINFOR
        CEMENTS
P8 1610 FORJ=ØTO1:FORK=ØTO2Ø:A
        =INT(RND(1)*K*3):PORL=
        1TO5:A=A+INT(RND(1)*21
        -8)
JR 1620 NEXTL:IFA<16THENA=Ø:GO
        TO1630
XF 1625 A=(A+K*8)AND254
HA 1630 FQ(K,J)=A:NEXTK,J
RG 1640 RETURN
EH 1700 REM ARMIES->MAP
SC 1710 FORJ=ØTO8:FORK=ØTO8
FX 1720 A=MAP(J,K,1):IFATHENA=
        A-1:GOSU81800
HH 1730 NEXTK,J
PH 1740 FORA=ØTO1:E=13+A*12:F=
        A*22:DX=2-4*A:D=Ø
PF 1750 FORJ=ØTO8:C=FQ(J,A):GO
        SUB1840
HP 1760 E=E+DX*2:IFJ>3THENF=F+
        DX:E=E-DX
XC 1770 NEXTJ,A
CA 1780 RETURN
QC 1800 8=MAP(J,K,Ø)
PH 1810 C=ARMY(8,Ø,A)
8K 1820 D=ARMY(8,6,A)
FJ 1830 E=(J-K+1Ø)*2-1:F=(13-J
        -K)*2+1: REM T&8 TO X/
        Y
SK 1840 XC=E:YC=P:GOSU81:IFATH
        ENPRINTP1$:GOTO1850
S8 1845 PRINTPØ$
MH 1850 IFC=ØTHENRETURN
DF 1860 POKEC5,6-2*A:XC=E:YC=F
        +A:GOSU81
CF 1870 QV=C:GOSU83:PRINTCHR$(
        18);Z$;CHR$(146)
RD 1880 IFDTHENF=F+1-A:G=1024+
        E+F*40:POKEG,43:POKEG+
        1,43:REM ++
XK 1890 RETURN
GC 1900 SW=Ø:E=NX(PN)-1:IFE<1T
        HENRETURN
SP 1910 FORJ=1TOE-1:IFARMY(J,Ø
        ,PN)>=1THEN1970
SE 1920 T=ARMY(J,4,PN):8=ARMY(
        J,5,PN):IFMAP(T,8,Ø)=J
        THENMAP(T,8,Ø)=Ø
8P 1930 FORK=JTOE:FORL=ØTO6:AR
        MY(K,L,PN)=ARMY(K+1,L,
        PN):ARMY(K+1,L,PN)=Ø:N
        EXTL
JQ 1940 T=ARMY(K,4,PN):8=ARMY(
        K,5,PN):MAP(T,8,Ø)=K
PH 1950 NEXTK
CA 1960 NX(PN)=NX(PN)-1:J=E:SW
        =1
FJ 1970 NEXTJ:IP SW THEN1900?
MG 2000 FORJ=1TOE:ARMY(J,Ø,PN)
        =ARMY(J,Ø,PN)+ARMY(J,2
        ,PN)
SD 2010 ARMY(J,2,PN)=ARMY(J,3,
        PN):ARMY(J,3,PN)=Ø
QS 2020 ARMY(J,6,PN)=Ø
PP 2030 NEXTJ:K=NX(1-PN):FOR J
        =1TOK:ARMY(J,6,1-PN)=Ø
        :NEXT
XF 2040 GOSU82400
MC 2050 IF8P>ØTHENFORJ=ØTO1:FO
        RK=1TOBP:A=BTL(K,J,Ø):
        AR(A,6,J)=AR(A,6,J)+1:
        NEXTK,J
CA 2060 RETURN
KG 2100 GOSU82400
BX 2110 A=NX(Ø):IPNX(1)>ATHENA
        =NX(1)
EX 2120 FORJ=ØTO1:FORK=1TOA:AR
        MY(K,6,J)=Ø:NEXTK,J
GP 2130 GOSU82050
SH 2140 RETURN
P8 2200 FORJ=ØTO1:A=1-J:8=NX(J
        )-1
DD 2210 FORK=1TO8
JQ 2220 IP ARMY(K,Ø,J)>=1 THEN
         2280
QD 2230 FQ(2,A)=FQ(2,A)+ARMY(K
        ,2,J)+ARMY(K,3,J)
XQ 2235 IP FQ(2,A)>255 THEN C=
        FQ(2,A)-255:FQ(3,A)=FQ
        (3,A)+C:FQ(2,A)=255
RM 2240 FQ(6,A)=FQ(6,A)+ARMY(K
        ,1,J)
JC 2245 IF FQ(6,A)>255 THEN C=
        FQ(6,A)-255:FQ(7,A)=FQ
        (7,A)+C:FQ(6,A)=255
XK 2250 IF MAP(ARMY(K,4,J),ARM
        Y(K,5,J),Ø)<>K THEN 22
        6Ø
DD 2252 IF MAP(ARMY(K,4,J),ARM
        Y(K,5,J),1)<>J+1 THEN
        {SPACE}226Ø
RR 2253 MAP(ARMY(K,4,J),ARMY(K
        ,5,J),Ø)=Ø
RB 2260 FORL=ØTO6:ARMY(K,L,J)=
        Ø:NEXTL
AS 2280 IF ARMY(K,6,J)>=1 THEN
         232Ø:REM EVACUATE INJ
        URED
AE 2290 FQ(4,J)=FQ(4,J)+ARMY(K
        ,1,J): ARMY(K,1,J)=Ø
DJ 2300 IF FQ(4,J)>255 THEN C=
        FQ(4,J)-255:FQ(5,J)=FQ
        (5,J)+C:FQ(4,J)=255
XP 2320 NEXTK,J:RETURN
SJ 2400 8P=Ø
KG 2410 FORJ=ØTO8:J1=(J-4)*(4-
        J>Ø):J2=8-(J>4)*(4-J):
        FORK=J1TOJ2
JE 2420 A=MAP(J,K,Ø)
HO 2430 R=MAP(J,K,1)
BG 2440 IF (A=Ø)OR(R=Ø) THEN24
        9Ø
KO 2450 IF ARMY(A,Ø,R-1)<1 THE
        N249Ø
DK 2460 T=J+1:8=K:GOSU8250Ø
HK 2470 B=B-1:GOSUB250Ø
8P 2480 T=T-1:GOSU8250Ø
RH 2490 NEXTK,J:RETURN
RR 2500 IF(T<Ø)OR(B<Ø)OR(T>8)O
        R(B>8)THEN RETURN
MA 2510 PA=MAP(T,8,Ø):IF PA=Ø
        {SPACE}THEN RETURN
FG 2520 IF MAP(T,8,1)=R THEN R
        ETURN
FX 2530 IF ARMY(PA,Ø,2-R)<1 TH
        EN RETURN
MA 2540 8P=8P+1:8TL(8P,R-1,Ø)=
        A:8TL(8P,2-R,Ø)=PA:RET
        URN
CF 2600 IF8P=ØTHENRETURN
PC 2610 FORJ=1TO8P
XM 2620 FORK=ØTO1:A=1-K
QX 2630 AN=BTL(J,K,Ø)
8P 2640 AS=ARMY(AN,Ø,K):HT=ARM
        Y(AN,6,K):CT=INT(AS/HT
        )+1
GH 2650 BTL(J,A,1)=INT(CT*KA+1
        )
JC 2660 8TL(J,A,2)=INT(CT*K8+1
        )
MB 2670 BTL(J,A,3)=INT(CT*KC+1
        )
XC 2680 NEXTK,J
MD 2700 FORJ=1TO8P:JØ=8TL(J,Ø,
        Ø):J1=8TL(J,1,Ø)
8R 2710 GOSU831ØØ
CG 2720 ARMY(JØ,Ø,Ø)=ARMY(JØ,Ø
        ,Ø)-A*8TL(J,Ø,1)
SK 2730 ARMY(J1,Ø,1)=ARMY(J1,Ø
        ,1)-8*8TL(J,1,1)
MS 2740 GOSU831ØØ
8C 2750 C=A*8TL(J,Ø,2):ARMY(JØ
        ,Ø,Ø)=ARMY(JØ,Ø,Ø)-C:A
        RMY(JØ,1,Ø)=ARMY(JØ,1,
        Ø)+C
FJ 2760 C=8*8TL(J,1,2):ARMY(J1
        ,Ø,1)=ARMY(J1,Ø,1)-C:A
        RMY(J1,1,1)=ARMY(J1,1,
        1)+C
QD 2770 GOSU831ØØ
AX 2780 C=A*8TL(J,Ø,3):ARMY(JØ
        ,Ø,Ø)=ARMY(JØ,Ø,Ø)-C:A
        RMY(JØ,3,Ø)=ARMY(JØ,3,
        Ø)+C
PF 2790 C=8*8TL(J,1,3):ARMY(J1
        ,Ø,1)=ARMY(J1,Ø,1)-C:A
        RMY(J1,3,1)=ARMY(J1,3,
        1)+C
XP 2800 NEXTJ
C8 2810 RETURN
GQ 2900 A=1-PN:B=Ø
GG 2910 FORJ=ØTO8:FORK=ØTO8
PA 292Ø IFMAP(J,K,1)=PN+1THEN8
        =B+1
KD 293Ø NEXTK,J
DM 2950 FQ(1,PN)=FQ(1,PN)+8
HG 2955 IFFO(1,PN)>255THENB=FQ
        (1,PN)-255:FQ(2,PN)=FQ
        (2,PN)+8:FQ(1,PN)=255
GQ 2960 T=4:8=PN*8
SJ 2970 IF MAP(T,8,Ø)<>Ø THEN
        {SPACE}RETURN
JK 2980 IP MAP(T,8,1)=PN+1 THE
        N PQ(Ø,A)=Ø:FO(1,A)=Ø:
        GOTO306Ø
XS 2990 J=NX(A):IFJ>31THENRETU
        RN
HR 3000 J1=FQ(Ø,A):IF J1<1 THE
        N306Ø
EP 3010 NX(A)=NX(A)+1
XE 3020 MAP(T,8,Ø)=J:MAP(T,8,1
        )=A+1
BK 3030 ARMY(J,Ø,A)=J1
OR 3040 FORK=1TO3:ARMY(J,K,A)=
        Ø:NEXTK
PM 3050 ARMY(J,4,A)=T:ARMY(J,5
        ,A)=B
KH 3060 FORK=ØTO19:FQ(K,A)=FQ(
        K+1,A):NEXTK
PK 3070 FQ(2Ø,A)=Ø
8A 3080 RETURN
FS 3100 A=Ø:FORM=1TO6:IFRND(1)
```

```
           <.5THENA=A+1
AH 3110 NEXTM:B=6-A:RETURN
RD 3200 REM WINDOW
CP 3210 Z=6:PRINT"{7 DOWN}"SPC
        (Z)"⁅7⁆⁅2 M⁆⁅2 L⁆{RVS}
        ⁅2 K⁆⁅J⁆⁅H⁆⁅G⁆ SCENARI
        O ⁅M⁆⁅N⁆⁅L⁆{OFF}⁅3 K⁆
        ⁅J⁆⁅2 G⁆":GOSUB 3310
XR 3220 PRINTSPC(Z)"⁅M⁆
        {2 SPACES}1> CAPTURE C
        APITAL/FAR{2 SPACES}
        ⁅G⁆"
DB 3230 PRINTSPC(Z)"⁅M⁆
        {2 SPACES}2> CAPTURE C
        APITAL/NEAR ⁅G⁆"
PF 3240 PRINTSPC(Z)"⁅M⁆
        {2 SPACES}3> OCCUPY
        {3 SPACES}8/12 CITIES
        {SPACE}⁅G⁆"
SF 3250 PRINTSPC(Z)"⁅M⁆
        {2 SPACES}4> CONTROL
        {2 SPACES}6/12 CITIES
        {SPACE}⁅G⁆"
XH 3260 PRINTSPC(Z)"⁅M⁆
        {2 SPACES}5> OCCUPY
        {2 SPACES}40/61 HEXES
        {2 SPACES}⁅G⁆":GOSUB33
        10
HX 3270 PRINTSPC(Z)"⁅2 M⁆⁅2 L⁆
        {RVS}⁅2 K⁆⁅J⁆⁅H⁆⁅G⁆
        {10 SPACES}⁅M⁆⁅N⁆⁅L⁆
        {OFF}⁅3 K⁆⁅J⁆⁅2 G⁆"
MK 3280 POKE198,0:WAIT198,1:GE
        TA$:GN=VAL(A$):IFGN<1O
        RGN>5THEN3280
QK 3290 XC=8:YC=8+GN:GOSUB1:PR
        INT"ZZ":FORTD=1TO1000:
        NEXT:PRINT"{HOME}"
PX 3300 RETURN
XX 3310 PRINTSPC(Z)"⁅M⁆
        {26 SPACES}⁅G⁆":RETURN
GS 3400 A=0:ON GN GOSUB 3430,3
        450,3480,3490,3500
JH 3410 IFA=0THENRETURN
QP 3415 QQ=A:EN$=C$:GOSUB600:G
        OSUB1710:C$=EN$:A=QQ
MJ 3420 PRINT"{HOME}PLAYER";A;
        " WINS":PRINTC$:PRINT"
        (PRESS ANY KEY)"
SE 3425 POKE198,0:WAIT198,1:RU
        N
BB 3430 IF MAP(CIT(2,0),CIT(2,
        1),1)=1THENA=2:C$="BLU
        E CAPTURED THE CAPITAL
        ":RETURN
CR 3440 GOTO3460
SG 3450 IF MAP(CIT(3,0),CIT(3,
        1),1)=1THENA=2:C$="BLU
        E CAPTURED THE CAPITAL
        ":RETURN
KB 3460 IF MAP(CIT(1,0),CIT(1,
        1),1)=2THENA=1:C$="VIO
        LET CAPTURED THE CAPIT
        AL"
FP 3470 RETURN
BC 3480 L=8:GOTO3500
JB 3490 L=6
FG 3500 C(1)=0:C(2)=0
PK 3510 FORJ=1TO12:T=CIT(J,0):
        B=CIT(J,1)
MB 3520 R=MAP(T,B,1):C(R)=C(R)
        +1
FB 3530 IF GN=4THEN AN=MAP(T,B
        ,0)
CX 3535 IFGN<>4 OR R<=0 THEN35
        40
HX 3537 IF AN=0 OR ARMY(AN,6,R
        -1)>0 THEN C(R)=C(R)-1
KJ 3540 NEXTJ
CB 3550 IF C(1)=> L THEN A=2:C
        $="BLUE HAS CAPTURED"+
        STR$(C(1))+" CITIES":R
        ETURN
QF 3560 IF C(2)=> L THEN A=1:C
        $="VIOLET HAS CAPTURED
        "+STR$(C(2))+" CITIES"
AB 3570 RETURN
RK 3580 C(1)=0:C(2)=0
PP 3590 FORJ=0TO8:FORK=0TO8
KM 3600 R=MAP(J,K,1):C(R)=C(R)
        +1
SR 3610 NEXTK,J
CE 3620 IF C(1)=>40THENA=2:C$=
        "BLUE OCCUPIES"+STR$(C
        (1))+" HEXES":RETURN
RK 3630 IF C(2)=>40THENA=1:C$=
        "VIOLET OCCUPIES"+STR$
        (C(2))+" HEXES"
DF 3640 RETURN
```

Program 3. Loader for 64 Hex War

```
AG 10 DV=8:Q$=CHR$(34)
DR 20 PRINT"{CLR}POKE 44,64:PO
      KE 16384,0:NEW"
XK 30 PRINT"{2 DOWN}LOAD"Q$"HE
      X WAR"Q$","DV
QA 40 POKE 198,3:POKE 631,19:P
      OKE 632,13:POKE 633,131
```

Program 4. Hex War For Atari 600XL, 800, 800XL, 1200XL and 130XE

Version by Kevin Mykytyn, Editorial Programmer

```
CK 0 PRINT "{CLEAR}":GOTO 9
KP 2 JOY$="QQQQ{D}{B}{C}Q
     {F}{H}{G}Q{E}{A}{,}":Z=
     STICK(0):J=ASC(JOY$(Z,Z
     ))+128-128*STRIG(0):RET
     URN
GH 3 IF QV>255 THEN QV=QV-25
     5:GOTO 3
ED 4 H$="0123456789ABCDEF":L
     V=INT(QV/16)+1:Z$=H$(LV
     ,LV):LV=QV-INT(QV/16)*1
     6+1:Z$(2,2)=H$(LV,LV):R
     ETURN
HH 9 GOSUB 4010:GOSUB 5000:G
     OSUB 6000
PC 30 DIM H$(16),Z$(6),JOY$(
      16),ARMY(31,14),BTL(64
      ,8),MAP(9,29),FQ(20,1)
      ,NX(1),C(2),P0$(8),P1$
      (8),C$(50),D1$(8)
HB 31 CN=12:DIM CIT(CN,1),D2
      $(8),T$(50)
LH 32 POKE 53248,0:HT=0:HB=0
      :J=0:K=0:CT=0:CB=0:J1=
      0:J2=0:A=0:B=0:C=0
LE 35 FOR A=0 TO 9:FOR B=0 T
      O 29:MAP(A,B)=0:NEXT B
      :NEXT A:FOR A=0 TO 31:
      FOR B=0 TO 14:ARMY(A,B
      )=0:NEXT B:NEXT A
LF 37 FOR A=0 TO 8:FOR B=0 T
      O 64:BTL(B,A)=0:NEXT B
      :NEXT A
GH 60 P0$="#${DOWN}{2 LEFT}
      {2 M}":P1$="{2 N}
      {DOWN}{2 LEFT})*"
PN 70 PN=1:ME=31:SC=704:C4=7
      12
AA 80 MM=3:REM MAX MOVES
JI 90 KA=1/48:KB=1/48:KC=1/3
      2
KD 260 RESTORE 270:FOR J=1 T
       O CN:FOR K=0 TO 1:REA
       D Z:CIT(J,K)=Z:NEXT K
       :MAP(CIT(J,0),CIT(J,1
       )+10*2)=1:NEXT J
AH 270 DATA 8,4,0,4,8,0,0,8,
       4,0,4,8
AI 280 DATA 5,5,3,3,6,3,2,5,
       5,2,3,6
EJ 300 GOSUB 3200:POSITION 1
       5,20:PRINT "SETTING U
       P"
HH 340 GOSUB 1500
CH 410 POKE 764,255
LD 420 GOSUB 1900:GOSUB 600:
       GOSUB 1710
CC 430 POKE C4,54+100*(1-PN)
       :GOSUB 800:POKE 53248
       ,0
KM 440 POKE C4,104:GOSUB 210
       0
FI 450 POKE C4,56:GOSUB 2600
BJ 460 POKE C4,0:GOSUB 2100
CD 470 POKE C4,8:GOSUB 2200
DH 480 GOSUB 2900
CH 490 POKE C4,7:GOSUB 3400
GM 500 PN=1-PN
BE 510 GOTO 410
IK 600 POKE C4,1:POKE SC,15:
       GRAPHICS 0:POSITION 0
       ,2:POKE 82,0:POKE 756
       ,CHBAS:POKE 559,62:PO
       KE 54279,CHBAS
IL 610 SETCOLOR 2,0,12:SETCO
       LOR 1,0,4:FOR J=1 TO
       5:POKE 85,13-2*J:FOR
       K=1 TO J+3:PRINT "-.
        ";:NEXT K:PRINT "-."
AL 620 POKE 752,1:POKE 85,12
       -2*J:FOR K=1 TO J+4:P
       RINT ",  +";:NEXT K:P
       RINT :NEXT J
DJ 630 FOR J=1 TO 5:POKE 85,
       J*2:FOR K=1 TO 10-J:P
       RINT ".  -";:NEXT K:P
       RINT
FE 640 POKE 85,J*2+1:FOR K=1
        TO 9-J:PRINT "+,  ";
       :NEXT K:PRINT "+,"
CE 650 NEXT J
PD 660 C$="{Q}{E}{DOWN}
       {2 LEFT}{Z}{C}":D1$="
       %&{DOWN}{2 LEFT}'(":D
       2$=":;{DOWN}{2 LEFT}<
       ="
FN 670 FOR J=1 TO 12:GOSUB 7
       10:NEXT J
JH 680 J=1:IF GN=1 THEN C$=D
       1$:GOSUB 710:J=2:C$=D
       2$:GOSUB 710
KD 690 IF GN=2 THEN C$=D1$:G
       OSUB 710:J=3:C$=D2$:G
       OSUB 710
LC 700 POSITION 0,0:RETURN
OF 710 K=CIT(J,0):L=CIT(J,1)
       :X=(K-L)*2+19:Y=(12-(
       K+L))*2+3
DH 715 POSITION X,Y:PRINT C$
       :RETURN
DJ 800 IF NX(PN)<2 THEN RETU
       RN
BH 805 HT=4:HB=4:GOSUB 1000:
       POKE BC,101
EH 810 MV=0:CT=0:CB=0:PK=0:K
       =0
OD 820 FOR J=1 TO NX(PN)-1:I
       F (ARMY(J,PN*7)>0) AN
       D (ARMY(J,6+PN*7)<1)
       THEN K=1:J=NX(PN)-1
LE 830 NEXT J:IF K=0 THEN RE
       TURN
PB 840 IF PEEK(764)=12 THEN
       RETURN
```

```
JK 850 GOSUB 2:IF J=Ø THEN 84Ø
OO 855 IF J>=128 THEN POKE 77,Ø:GOTO 11ØØ
HL 857 IF (J/2)-INT(J/2)=Ø THEN 84Ø
DP 860 J=(J-1)/2:IF (J/2)-INT(J/2) THEN B1=HB+J-2:T1=HT:GOTO 870
EI 865 T1=HT+1-J:B1=HB
CI 870 IF (T1<Ø) OR (T1>8) THEN 84Ø
AF 880 IF (B1<Ø) OR (B1>8) THEN 84Ø
HF 890 S1=T1+B1:IF (S1<4) OR (S1>12) THEN 84Ø
EK 895 HB=B1:HT=T1:GOSUB 1ØØØ:GOSUB 94Ø
JG 9ØØ QN=MAP(HT,HB):IF QN=Ø THEN 84Ø
MI 9Ø5 Q1=MAP(HT,HB+1Ø)-1
MM 91Ø POSITION Ø,Ø:PRINT "";QN:PRINT "{5 M}"
MF 92Ø FOR J=Ø TO 3:PRINT "";ARMY(QN,J+Q1*7):NEXT J
HO 93Ø GOTO 84Ø
FI 94Ø POSITION Ø,Ø:FOR Z=1 TO 6:PRINT "{5 SPACES}":NEXT Z:RETURN
JF 1ØØØ SX=12Ø+8*(HT-HB):SY=248-16*(HT+HB):POKE 53248,SX:POKE 2Ø3,SY:Z=USR(1536)
HB 1Ø2Ø IF MAP(HT,HB+2Ø)=1 THEN POKE SC,72:RETURN
BB 1Ø3Ø POKE SC,1Ø1:RETURN
CC 11ØØ GOSUB 2:IF J<>Ø THEN 11ØØ
EN 111Ø IF PK=1 THEN 12ØØ
FH 112Ø IF ((MAP(HT,HB+1Ø)<>PN+1) OR (MAP(HT,HB)=Ø)) THEN GOTO 81Ø
PC 113Ø AN=MAP(HT,HB):IF ARMY(AN,6+PN*7)<>Ø THEN 81Ø
FA 114Ø PK=1:CT=HT:CB=HB:CS=ARMY(AN,PN*7)
PA 115Ø SOUND 1,1ØØ,1Ø,1Ø:FOR A=1 TO 3Ø:NEXT A:SOUND 1,Ø,Ø,Ø
JM 116Ø GOTO 84Ø
MM 12ØØ J=((HT=CT) AND (HB=CB))
JJ 121Ø IF J AND (MV=Ø) THEN 81Ø
MJ 122Ø IF J AND (MV>Ø) THEN 142Ø
MB 1225 AS=ARMY(MAP(HT,HB),PN*7)
LN 123Ø IF ((AS>ME) AND (CS>ME)) OR ((MAP(HT,HB+1Ø)-1=1-PN) AND (AS>Ø)) THEN 84Ø
IM 1235 DT=ABS(CT-HT):DB=ABS(CB-HB)
DE 124Ø TL=DB+DT:IF NOT ((TL=1) OR ((CT+CB=HT+HB) AND (DT=1))) THEN 84Ø
MH 125Ø MG=MAP(HT,HB):IF MG=Ø THEN 13ØØ
DH 126Ø FOR J=Ø TO 3:ARMY(MG,J+PN*7)=ARMY(MG,J+PN*7)+ARMY(AN,J+PN*7):ARMY(AN,J+PN*7)=Ø:NEXT J
AA 127Ø ARMY(MG,6+PN*7)=1:MAP(CT,CB)=Ø
JB 128Ø CS=ARMY(MG,PN*7):AN=MG:MV=MM+1
MB 129Ø GOTO 138Ø
MB 13ØØ NB=MAP(HT,HB+1Ø)-1:MV=MV+1:IF (NB<>PN) THEN MV=MV+1
KJ 131Ø MAP(CT,CB)=Ø
DM 132Ø MAP(HT,HB)=AN:MAP(HT,HB+1Ø)=PN+1:ARMY(AN,4+PN*7)=HT:ARMY(AN,5+PN*7)=HB
FB 1322 IF MV>=MM THEN ARMY(AN,6+PN*7)=1
JA 1325 K=Ø:FOR J=-1 TO 1 STEP 2:J1=HT+J:J2=HB+J:J3=HB-J:IF (J1<Ø) OR (J1>8) THEN 134Ø
KM 1327 IF (MAP(J1,HB)<=Ø) THEN 134Ø
DE 133Ø IF (MAP(J1,HB+1Ø)=2-PN) THEN K=1:J=1:GOTO 136Ø
GM 134Ø IF (J2<Ø) OR (J2>8) THEN 135Ø
OK 1342 IF (MAP(HT,J2)>Ø) THEN IF (MAP(HT,J2+1Ø)=2-PN) THEN K=1:J=1:GOTO 136Ø
CM 135Ø IF (J3<Ø) OR (J3>8) OR (J1<Ø) OR (J1>8) THEN 136Ø
GO 1352 IF (MAP(J1,J3)>Ø) THEN IF (MAP(J1,J3+1Ø)=2-PN) THEN K=1:J=1
FD 136Ø NEXT J
GF 137Ø IF K=1 THEN ARMY(AN,6+PN*7)=1:MV=MM+1
OC 138Ø A=PN:J=CT:K=CB:C=Ø:D=Ø:GOSUB 183Ø
HC 139Ø J=HT:K=HB:C=CS:D=ARMY(AN,6+PN*7):GOSUB 183Ø
LL 14ØØ CT=HT:CB=HB
JJ 141Ø IF MV<MM THEN 84Ø
EF 142Ø ARMY(AN,6+PN*7)=1:J=HT:K=HB:C=CS:D=1:GOSUB 183Ø
JK 143Ø GOTO 81Ø
HC 15ØØ RESTORE 154Ø:FOR J=Ø TO 1:NX(J)=5:FOR K=1 TO 4:READ A,B,C
KG 151Ø ARMY(K,J*7)=A:ARMY(K,4+J*7)=B:ARMY(K,5+J*7)=C:MAP(B,C)=K:MAP(B,C+1Ø)=J+1
BF 152Ø NEXT K:NEXT J
BL 154Ø DATA 64,2,8,64,3,7,64,5,6,64,6,6
AG 155Ø DATA 64,2,2,64,3,2,64,5,1,64,6,Ø
EI 16ØØ REM REINFORCE
ML 161Ø FOR J=Ø TO 1:FOR K=Ø TO 2Ø:A=INT(RND(1)*K*3):A=INT(RND(1)*K*3):FOR L=1 TO 5:A=A+INT(RND(1)*21-8)
BL 162Ø NEXT L:IF A<16 THEN A=Ø:GOTO 163Ø
IJ 1625 Z=A+K*8:A=INT(Z/2)*2
HI 163Ø FQ(K,J)=A:NEXT K:NEXT J
KL 164Ø RETURN
IJ 17ØØ REM ARMIES>MAP
PG 171Ø FOR J=Ø TO 8:FOR K=Ø TO 8
JM 172Ø A=MAP(J,K+1Ø):IF A THEN A=A-1:GOSUB 18ØØ
BI 173Ø NEXT K:NEXT J
MB 174Ø FOR A=Ø TO 1:E=13+A*12:F=A*22:DX=2-4*A:D=Ø
CG 175Ø FOR J=Ø TO 8:C=FQ(J,A):GOSUB 184Ø
MF 176Ø E=E+DX*2:IF J>3 THEN F=F+DX:E=E-DX
BC 177Ø NEXT J:NEXT A
LA 178Ø RETURN
DI 18ØØ B=MAP(J,K)
DE 181Ø C=ARMY(B,A*7)
EH 182Ø D=ARMY(B,6+A*7)
MA 183Ø E=(J-K+1Ø)*2-1:F=(13-J-K)*2+1
OA 184Ø POSITION E,F:IF A THEN PRINT P1$;:GOTO 185Ø
DD 1845 PRINT PØ$;
BM 185Ø IF C=Ø THEN RETURN
MF 186Ø POSITION E,F+1-A
GE 187Ø QV=C:GOSUB 3:PRINT Z$;
LP 188Ø IF D THEN POSITION E,F+A:PRINT CHR$(63-A);CHR$(63-A)
LC 189Ø RETURN
BK 19ØØ SW=Ø:E=NX(PN)-1:IF E<1 THEN RETURN
CE 191Ø FOR J=1 TO E-1:IF ARMY(J,PN*7)>=1 THEN 197Ø
KB 192Ø T=ARMY(J,4+7*PN):B=ARMY(J,5+PN*7):IF MAP(T,B)=J THEN MAP(T,B)=Ø
CB 193Ø FOR K=J TO E:FOR L=Ø TO 6:ARMY(K,L+PN*7)=ARMY(K+1,L+PN*7):ARMY(K+1,L+PN*7)=Ø:NEXT L
IK 194Ø T=ARMY(K,4+PN*7):B=ARMY(K,5+PN*7):MAP(T,B)=K
FJ 195Ø NEXT K
ON 196Ø NX(PN)=NX(PN)-1:J=E:SW=1
MG 197Ø NEXT J:IF SW THEN 19ØØ
EF 2ØØØ FOR J=1 TO E:ARMY(J,PN*7)=ARMY(J,PN*7)+ARMY(J,2+PN*7)
LN 2Ø1Ø ARMY(J,2+PN*7)=ARMY(J,3+PN*7):ARMY(J,3+PN*7)=Ø
JB 2Ø2Ø ARMY(J,6+PN*7)=Ø
ED 2Ø3Ø NEXT J:K=NX(1-PN):FOR J=1 TO K:ARMY(J,6+7*(1-PN))=Ø:NEXT J
AM 2Ø4Ø GOSUB 24ØØ
BC 2Ø5Ø IF BP>Ø THEN FOR J=Ø TO 1:FOR K=1 TO BP:A=BTL(K,J):ARMY(A,6+J*7)=ARMY(A,6+J*7)+1:NEXT K:NEXT J
KI 2Ø6Ø RETURN
AJ 21ØØ GOSUB 24ØØ
KD 211Ø A=NX(Ø):IF NX(1)>A THEN A=NX(1)
DA 212Ø FOR J=Ø TO 1:FOR K=1 TO A:ARMY(K,6+J*7)=Ø:NEXT K:NEXT J
AM 213Ø GOSUB 2Ø5Ø
KH 214Ø RETURN
OO 22ØØ FOR J=Ø TO 1:A=1-J:B=NX(J)-1
EK 221Ø FOR K=1 TO B
KI 222Ø IF ARMY(K,J*7)>=1 THEN 228Ø
DD 223Ø FQ(2,A)=FQ(2,A)+ARMY(K,2+J*7)+ARMY(K,3+J*7):IF FQ(2,A)>255 THEN C=FQ(2,A)-255:FQ(3,A)=FQ(3,A)+C:FQ(2,A)=255
BK 224Ø FQ(6,A)=FQ(6,A)+ARMY(K,1+J*7):IF FQ(6,A)>255 THEN C=FQ(6,A)-255:FQ(7,A)=FQ(7,A)+C:FQ(6,A)=255
KF 225Ø IF (MAP(ARMY(K,4+J*7
```

```
),ARMY(K,5+J*7))=K) AND (MAP(ARMY(K,4+J*7),ARMY(K,5+J*7)+10))=J+1 THEN 2255
MN 2251 GOTO 2260
KN 2255 MAP(ARMY(K,4+J*7),ARMY(K,5+J*7))=0
NC 2260 FOR L=0 TO 6:ARMY(K,L+J*7)=0:NEXT L
AK 2280 IF ARMY(K,6+J*7)>=1 THEN 2320
BA 2290 FQ(4,J)=FQ(4,J)+ARMY(K,1+J*7):ARMY(K,1+J*7)=0
HK 2300 IF FQ(4,J)>255 THEN C=FQ(4,J)-255:FQ(5,J)=FQ(5,J)+C:FQ(4,J)=255
CD 2320 NEXT K:NEXT J:RETURN
MF 2400 BP=0
LL 2410 FOR J=0 TO 8:J1=(J-4)*-(4-J>0):J2=8+(J>4)*(4-J):FOR K=J1 TO J2
DG 2420 A=MAP(J,K)
ME 2430 R=MAP(J,K+10)
BF 2440 IF A=0 OR R=0 THEN 2490
BD 2450 IF ARMY(A,(R-1)*7)<1 THEN 2490
II 2460 T=J+1:B=K:GOSUB 2500
BN 2470 B=B-1:GOSUB 2500
JC 2480 T=T-1:GOSUB 2500
DE 2490 NEXT K:NEXT J:RETURN
DI 2500 IF T<0 OR B<0 OR T>8 OR B>8 THEN RETURN
FN 2510 PA=MAP(T,B):IF PA=0 THEN RETURN
HD 2520 IF MAP(T,B+10)=R THEN RETURN
HP 2530 IF ARMY(PA,(2-R)*7)<1 THEN RETURN
KF 2540 BP=BP+1:BTL(BP,R-1)=A:BTL(BP,2-R)=PA:RETURN
BF 2600 IF BP=0 THEN RETURN
JN 2610 FOR J=1 TO BP
JD 2620 FOR K=0 TO 1:A=1-K
IL 2630 AN=BTL(J,K)
PI 2640 AS=ARMY(AN,K*7):HT=ARMY(AN,6+K*7):CT=INT(AS/HT)+1
DB 2650 BTL(J,A+2)=INT(CT*KA+1)
DK 2660 BTL(J,A+4)=INT(CT*KB+1)
DD 2670 BTL(J,A+6)=INT(CT*KC+1)
BK 2680 NEXT K:NEXT J
DD 2700 FOR J=1 TO BP:J0=BTL(J,0):J1=BTL(J,1)
AD 2710 GOSUB 3100
DL 2720 ARMY(J0,0)=ARMY(J0,0)-A*BTL(J,2)
ED 2730 ARMY(J1,7)=ARMY(J1,7)-B*BTL(J,3)
BB 2740 GOSUB 3100
DE 2750 C=A*BTL(J,4):ARMY(J0,0)=ARMY(J0,0)-C:ARMY(J0,1)=ARMY(J0,1)+C
AH 2760 C=B*BTL(J,5):ARMY(J1,7)=ARMY(J1,7)-C:ARMY(J1,8)=ARMY(J1,8)+C
BE 2770 GOSUB 3100
DM 2780 C=A*BTL(J,6):ARMY(J0,0)=ARMY(J0,0)-C:ARMY(J0,3)=ARMY(J0,3)+C
FD 2790 C=B*BTL(J,7):ARMY(J1,7)=ARMY(J1,7)-C:ARMY(J1,10)=ARMY(J1,10)+C
FD 2800 NEXT J
KL 2810 RETURN
CD 2900 A=1-PN:B=0
PJ 2910 FOR J=0 TO 8:FOR K=0 TO 8
FL 2920 IF MAP(J,K+10)=PN+1 THEN B=B+1
BL 2930 NEXT K:NEXT J
EA 2950 FQ(1,PN)=FQ(1,PN)+B
BN 2955 IF FQ(1,PN)>255 THEN B=FQ(1,PN)-255:FQ(2,PN)=FQ(2,PN)+B:FQ(1,PN)=255
EP 2960 T=4:B=PN*8
AL 2970 IF MAP(T,B)<>0 THEN RETURN
KA 2980 IF MAP(T,B+10)=PN+1 THEN FQ(0,A)=0:FQ(1,A)=0:GOTO 3060
FH 2990 J=NX(A):IF J>31 THEN RETURN
KJ 3000 J1=FQ(0,A):IF J1<1 THEN 3060
MM 3010 NX(A)=NX(A)+1
MD 3020 MAP(T,B)=J:MAP(T,B+10)=A+1
CA 3030 ARMY(J,A*7)=J1
MA 3040 FOR K=1 TO 3:ARMY(J,K+A*7)=0:NEXT K
BF 3050 ARMY(J,4+A*7)=T:ARMY(J,5+7*A)=B
EM 3060 FOR K=0 TO 19:FQ(K,A)=FQ(K+1,A):NEXT K
OO 3070 FQ(20,A)=0
KL 3080 RETURN
BP 3100 A=0:FOR M=1 TO 6:IF RND(1)<0.5 THEN A=A+1
MI 3110 NEXT M:B=6-A:RETURN
IB 3200 REM WINDOW
DB 3210 GRAPHICS 0:CLOSE #4:OPEN #4,12,4,"K:":Z=7:POSITION 16,4:PRINT "SCENARIO":POKE 752,1
FN 3220 POSITION 9,8:PRINT "1. CAPTURE CAPITAL/FAR"
NF 3230 POSITION 9,10:PRINT "2. CAPTURE CAPITAL/NEAR"
MA 3240 POSITION 9,12:PRINT "3. OCCUPY 8/12 CITIES"
CA 3250 POSITION 9,14:PRINT "4. CONTROL 6/12 CITIES"
ME 3260 POSITION 9,16:PRINT "5. OCCUPY 40/61 HEXES"
BF 3280 GET #4,A:IF CHR$(A)<"1" OR CHR$(A)>"5" THEN 3280
JG 3290 GN=VAL(CHR$(A)):POSITION 7,6+GN*2:PRINT "{2 .}"
KG 3300 RETURN
BB 3400 A=0:ON GN GOSUB 3430,3450,3480,3490,3580
BE 3410 IF A=0 THEN RETURN
BP 3415 IF A<>0 THEN T$=C$:QQ=A:GOSUB 600:GOSUB 1710:C$=T$:A=QQ
BD 3420 GOSUB 940:POSITION 0,0:PRINT "PLAYER ";A;" WINS":PRINT C$
MM 3422 PRINT "PRESS ANY KEY":POKE 764,255
EI 3424 IF PEEK(764)=255 THEN 3424
MP 3426 POKE C4,15:GOTO 32
KH 3430 IF MAP(CIT(2,0),CIT(2,1)+10)=1 THEN A=2:C$="BLUE CAPTURED THE CAPITAL":RETURN
NB 3440 GOTO 3460
KL 3450 IF MAP(CIT(3,0),CIT(3,1)+10)=1 THEN A=2:C$="BLUE CAPTURED THE CAPITAL":RETURN
EB 3460 IF MAP(CIT(1,0),CIT(1,1)+10)=2 THEN A=1:C$="RED CAPTURED THE CAPITAL"
KD 3470 RETURN
ML 3480 L=8:GOTO 3500
IP 3490 L=6
BK 3500 C(1)=0:C(2)=0
KB 3510 FOR J=1 TO 12:T=CIT(J,0):B=CIT(J,1)
HF 3520 R=MAP(T,B+10):C(R)=C(R)+1
ND 3530 IF GN=4 THEN AN=MAP(T,B):IF R>0 THEN IF (AN=0) OR (ARMY(AN,6+(R-1)*7)>0) THEN C(R)=C(R)-1
FF 3540 NEXT J
EF 3550 IF C(1)>=L THEN A=2:C$="BLUE HAS CAPTURED "
PJ 3560 IF C(2)>=L THEN A=1:C$="RED HAS CAPTURED "
JF 3565 IF A THEN Z=LEN(STR$(C(3-A))):C$(LEN(C$)+1,LEN(C$)+Z)=STR$(C(3-A)):C$(LEN(C$)+1,LEN(C$)+7)=" CITIES"
KP 3570 RETURN
BP 3580 C(1)=0:C(2)=0
PD 3590 FOR J=0 TO 8:FOR K=0 TO 8
HD 3600 R=MAP(J,K+10):C(R)=C(R)+1
BH 3610 NEXT K:NEXT J
IC 3620 IF C(1)>=40 THEN A=2:C$="BLUE OCCUPIES "
DB 3630 IF C(2)>=40 THEN A=1:C$="RED OCCUPIES "
ED 3635 IF A THEN Z=LEN(STR$(C(3-A))):C$(LEN(C$)+1,LEN(C$)+Z)=STR$(C(3-A)):C$(LEN(C$)+1,LEN(C$)+6)=" HEXES"
KM 3640 RETURN
DD 4010 CHBAS=PEEK(106)-8:POKE 106,CHBAS:GRAPHICS 0:CHSET=CHBAS*256
MD 4020 POKE 752,1:POSITION 14,10:PRINT "PLEASE WAIT":FOR A=0 TO 1023:POKE CHSET+A,PEEK(57344+A):NEXT A
KM 4070 RESTORE 4080:FOR A=CHSET+24 TO CHSET+119:READ B:POKE A,B:NEXT A
EK 4075 RESTORE 4090:FOR A=CHSET+208 TO CHSET+239:READ B:POKE A,B/2:NEXT A
LJ 4077 RESTORE 4140:FOR A=CHSET+240 TO CHSET+247:READ B:POKE A+8,B:POKE A,B*2:NEXT A
PG 4078 FOR A=CHSET+24 TO CHSET+71:POKE A,PEEK(A)/2:NEXT A:FOR A=CHSET+208 TO CHSET+239:POKE A,PEEK(A)*2:NEXT A
LF 4079 FOR A=CHSET+72 TO CHSET+87:POKE A,PEEK(A)*2:NEXT A:RETURN
BD 4080 DATA 2,2,10,10,42,42
```

```
            ,170,170,128,128,160
            ,160,16B,168,170,170
DD 4090 OATA 10,10,10,10,10,
        170,170,170,160,160,
        160,160,160,170,170,
        170
HL 4100 OATA 170,170,170,10,
        10,10,10,10,170,170,
        170,160,160,160,160,
        160
DG 4110 OATA 85,85,21,21,5,5
        ,1,1,85,85,B4,84,B0,
        B0,64,64
KH 4120 OATA 240,240,63,15,3
        ,3,3,3,15,15,252,240
        ,192,192,192,192
KI 4130 DATA 3,3,3,3,15,63,2
        40,240,192,192,192,1
        92,240,252,15,15
EI 4140 OATA 20,20,B5,85,85,
        20,20,20
CC 5000 POKE 623,1:POKE 5327
        7,3:POKE 704,15:POKE
         204,CHBAS+4:POKE 53
        256,1:RETURN
FD 6000 RESTORE 6000:FOR Z=1
        536 TO 1580:READ B:P
        OKE Z,B:NEXT Z:RETUR
        N
FM 6100 DATA 165,204,133,207
        ,169,0,133,206,168,1
        45,206,136,20B,251,1
        64,203,162,15,1B9,29
        ,6,145,206,200,202,1
        6
FL 6110 DATA 247,104,96,255,
        129,129,129,129,129,
        129,129,129,129,129,
        129,129,129,129,255
```

Program 5. Hex War For Apple II

Version by Tim Victor, Editorial Programmer

```
47 10 LOHEH: 24576
B7 20 DIH ARMY(31,6,1),BTL(64,1,
      3),MAP(9,9,2),FQ(20,1),NX(
      1),C(2)
A2 30 CN = 12: DIH CITY(CN,1)
5D 40 PN = 1:HE = 31:HM = 3:KA =
       1 / 4B:KB = 1 / 4B:KC = 1
       / 32
7E 50 FOR J = 1 TO CN: FOR K = 0
       TO 1: READ CITY(J,K): NEX
      T K:MAP(CITY(J,0),CITY(J,1
      ),2) = 1: NEXT J
1F 60 GOSUB 1950
D2 70 GOSUB B50
97 B0 GOSUB 2330
F5 90 POKE 6,0: POKE 7,129
F4 100 IF PEEK (190 * 256) = 76
        THEN 120
C9 110 POKE 54,0: POKE 55,3: CAL
       L 1002: GOTO 130
8B 120 PRINT CHR$ (4);"PR#A76B"
EB 130 A = FRE (0): GOSUB 1110:
       GOSUB 220: GOSUB 930
6F 140 GOSUB 340: VTAB 37: HTAB
       37: PRINT "  ";
DF 150 GOSUB 1250
E3 160 GOSUB 1540
E3 170 GOSUB 1250
F5 1B0 GOSUB 1290
F1 190 GOSUB 1750
CF 200 GOSUB 2040
5B 210 PN = 1 - PN: GOTO 130
23 220 HOHE : HGR2 : VTAB 3: FOR
        J = 1 TO 5: HTAB 13 - 2
       * J: FOR K = 1 TO J + 3:
       PRINT "%&  ";: NEXT K: PR
       INT "%&"
B4 230 HTAB 12 - 2 * J: FOR K =
       1 TO J + 4: PRINT "$  #";
       : NEXT K: PRINT : NEXT J
4E 240 FOR J = 1 TO 5: HTAB 2 *
       J: FOR K = 1 TO 10 - J: P
       RINT "&  %";: NEXT K: PRI
       NT
32 250 HTAB 2 * J + 1: FOR K = 1
        TO 9 - J: PRINT "#$  ";:
        NEXT K: PRINT "#$": NEXT
        J
7E 260 FOR J = 0 TO B:H1 = 62 +
       14 * ((J - B) * (J > 4) -
        J * (J < 5)):V1 = 27 + 1
       6 * J
F5 270 H2 = 265 - H1: HCOLOR= 7:
        HPLOT H1,V1 TO H1,V1 + 9
       : HPLOT H2,V1 TO H2,V1 +
       9
6F 2B0 NEXT J
2E 290 C1$ = "OP":C2$ = "QR": FO
       R J = 1 TO 12: GOSUB 330:
        NEXT
34 300 C1$ = "ST":C2$ = "UV":J =
        1: IF GN = 1 THEN GOSUB
       330:J = 2:C1$ = "WX":C2$
       = "YZ": GOBUB 330
2B 310 IF GN = 2 THEN GOSUB 330:
       C1$ = "WX":C2$ = "YZ":J =
        3: GOSUB 330
17 320 RETURN
99 330 K = CITY(J,0):L = CITY(J,
       1):X = (K - L) * 2 + 19:Y
        = (12 - (K + L)) * 2 + 4
       : VTAB Y: HTAB X: PRINT C
       1$: HTAB X: PRINT C2$: RE
       TURN
96 340 HT = 4:HB = 4: IF NX(PN)
       < 2 THEN RETURN
56 350 MV = 0:CT = 0:CB = 0:PK =
        0
2A 355 VTAB 21: HTAB 37: PRINT M
       ID$ ("<=>?",PN * 2 + 1,2)
       ;
5B 360 K = 0: FOR J = 1 TO NX(PN
       ) - 1: IF ARMY(J,0,PN) >
       0 AND ARMY(J,6,PN) < 1 TH
       EN K = 1:J = NX(PN) - 1
C2 370 NEXT J: IF K = 0 THEN RET
       URN
C7 3B0 HC = 4 * (MAP(HT,HB,1) <
       2): HCOLOR= 3 + HC: GOSUB
        B20: GET A$: HCOLOR= HC:
        GOSUB B20
97 390 IF A$ = CHR$ (13) THEN RE
       TURN
19 400 IF A$ = CHR$ (3) THEN STO
       P
72 410 IF A$ < > " " THEN 710
E9 420 IF PK = 1 THEN 460
9C 430 IF HAP(HT,HB,1) < > PN +
       1 OR MAP(HT,HB,0) = 0 THE
       N 350
72 440 AN = HAP(HT,HB,0): IF ARH
       Y(AN,6,PN) < > 0 THEN 350
23 450 PRINT CHR$ (7);:PK = 1:CT
        = HT:CB = HB:CS = ARMY(A
       N,0,PN): GOTO 3B0
C4 460 IF (HT < > CT) OR HB < >
       CB THEN 490
A6 470 IF HV > 0 THEN 690
A3 4B0 GOTO 350
D6 490 AS = ARHY(HAP(HT,HB,0),0,
       PN): IF AS > ME AND CS >
       ME OR MAP(HT,HB,1) = 2 -
       PN AND AS > 0 THEN 3B0
8E 500 OT = ABS (CT - HT):DB = A
       BS (CB - HB):TL = OB + OT
       : IF TL < > 1 AND (CT + C
       B < > HT + HB OR OT < > 1
       ) THEN 3B0
33 510 MG = HAP(HT,HB,0): IF MG
        = 0 THEN 560
FA 520 FOR J = 0 TO 3:ARMY(MG,J,
       PN) = ARMY(MG,J,PN) + ARM
       Y(AN,J,PN):ARMY(AN,J,PN)
       = 0: NEXT J
ED 530 ARMY(MG,6,PN) = 1:MAP(CT,
       CB,0) = 0
ED 540 CS = ARMY(MG,0,PN):AN = M
       G:HV = MM + 1
2B 550 GOTO 650
43 560 NB = MAP(HT,HB,1) - 1:MV
       = MV + 1: IF NB < > PN TH
       EN MV = MV + 1
69 570 MAP(CT,CB,0) = 0:MAP(HT,H
       B,0) = AN:MAP(HT,HB,1) =
       PN + 1:ARMY(AN,4,PN) = HT
       :ARMY(AN,5,PN) = HB: IF M
       V > = MH THEN ARMY(AN,6,P
       N) = 1
B1 5B0 K = 0: FOR J = - 1 TO 1 S
       TEP 2:J1 = HT + J:J2 = HB
        + J:J3 = HB - J: IF J1 <
        0 OR J1 > B THEN 600
A3 590 IF MAP(J1,HB,0) > 0 AND M
       AP(J1,HB,1) = 2 - PN THEN
        K = 1:J = 1: GOTO 640
8E 600 IF J2 < 0 OR J2 > B THEN
       620
BB 610 IF MAP(HT,J2,0) > 0 AND H
       AP(HT,J2,1) = 2 - PN THEN
        K = 1:J = 1: GOTO 640
BB 620 IF J3 < 0 OR J3 > B OR J1
        < 0 OR J1 > B THEN 640
2D 630 IF MAP(J1,J3,0) > 0 AND M
       AP(J1,J3,1) = 2 - PN THEN
        K = 1:J = 1
41 640 NEXT J: IF K = 1 THEN ARH
       Y(AN,6,PN) = 1:MV = MM +
       1
36 650 A = PN:J = CT:K = CB:C =
       0:O = 0: GOSUB 1040
6B 660 J = HT:K = HB:C = CS:O =
       ARMY(AN,6,PN): GOSUB 1040
CB 670 CT = HT:CB = HB
A6 6B0 IF MV < HH THEN 3B0
3E 690 ARHY(AN,6,PN) = 1:J = HT:
       K = HB:C = CS:D = 1: GOSU
       B 1040
96 700 GOTO 350
C7 710 IF A$ = "," THEN B1 = HB
       + 1:T1 = HT
D3 720 IF A$ = CHR$ (21) THEN B1
        = HB - 1:T1 = HT
37 730 IF A$ = CHR$ (B) THEN T1
       = HT - 1:B1 = HB
D1 740 IF A$ = "." THEN T1 = HT
       + 1:B1 = HB
17 750 IF T1 < 0 OR T1 > B OR B1
        < 0 OR B1 > B THEN 3B0
BB 760 S1 = T1 + B1: IF S1 < 4 O
       R S1 > 12 THEN 3B0
B4 770 HB = B1:HT = T1:QN = HAP(
       HT,HB,0): IF QN = 0 THEN
       3B0
C1 7B0 Q1 = MAP(HT,HB,1) - 1
D4 790 VTAB 1: HTAB 1:A$ = STR$
       (QN): PRINT SPC( 4 - LEN
       (A$));A$: PRINT "______"
7A B00 FOR J = 0 TO 3:A$ = STR$
       (ARMY(QN,J,Q1)): PRINT SP
       C( 4 - LEN (A$));A$: NEXT
        J
9C B10 GOTO 3B0
D2 B20 HH = 120 + (HT - HB) * 14
       :HV = 219 - (HT + HB) * 1
       6
B2 B30 HPLOT HH,HV TO HH + 12,HV
        - 6 TO HH + 25,HV + 1 TO
        HH + 25,HV + 9 TO HH + 1
       3,HV + 15 TO HH,HV + B TO
        HH,HV
2B B40 RETURN
```

```
DC 850 FOR J = Ø TO 1:NX(J) = 5:
       FOR K = 1 TO 4: READ A,B
       ,C
BC 860 ARMY(K,Ø,J) = A:ARMY(K,4,
       J) = B:ARMY(K,5,J) = C:MA
       P(B,C,Ø) = K:MAP(B,C,1) =
        J + 1
36 870 NEXT K,J
45 880 FOR J = Ø TO 1: FOR K = Ø
        TO 20
55 890 A = INT ( RND (1) * K * 3
       ): FOR L = 1 TO 5:A = A +
        INT ( RND (1) * 21 - 8):
        NEXT L: IF A < 16 THEN A
        = Ø: GOTO 910
F2 900 A = (A + K * B):A = 2 * I
       NT (A / 2)
8D 910 FQ(K,J) = A: NEXT K,J
1C 920 RETURN
B9 930 FOR J = Ø TO 8: FOR K = Ø
        TO 8
A2 940 A = MAP(J,K,1): IF (A) TH
       EN A = A - 1: GOSUB 1010
33 950 NEXT K,J
32 960 FOR A = Ø TO 1:E = 13 + A
        * 12:F = A * 22 + 1:OX =
        2 - 4 * A:D = Ø
2A 970 FOR J = Ø TO 8:C = FQ(J,A
       ): VTAB F: HTAB E: GOSUB
        1050
DC 980 E = E + OX * 2: IF J > 3
        THEN F = F + OX:E = E - O
       X
AB 990 NEXT J,A
D1 1000 RETURN
9D 1010 B = MAP(J,K,Ø)
C2 1020 C = ARMY(B,Ø,A)
D6 1030 D = ARMY(B,6,A)
3F 1040 GOSUB 1100
A4 1050 PRINT MID$ ("<=*+",A * 2
        + 1,2)
BB 1060 HTAB E: PRINT MID$ ("()>
       ?",A * 2 + 1,2);: IF C =
        Ø THEN 1090
67 1070 H$ = "":X = C: FOR L = Ø
        TO 1:T = X:X = INT (X /
        16):T = T - 16 * X:H$ =
        CHR$ (T + 48 + 7 * (T >
        9)) + H$: NEXT
25 1080 VTAB F + 1 - A: HTAB E:
        PRINT H$;: IF D THEN VTA
        B F + A: HTAB E: PRINT M
        ID$ (",-./",A * 2 + 1,2)
        ;
93 1090 NORMAL : RETURN
6B 1100 E = (J - K + 10) * 2 - 1
        :F = (13 - J - K) * 2 +
        2: HTAB E: VTAB F: RETUR
        N
C8 1110 SW = Ø:E = NX(PN) - 1: I
        F E < 1 THEN RETURN
11 1120 FOR J = 1 TO E - 1: IF A
        RMY(J,Ø,PN) > = Ø THEN 1
        170
D8 1130 T = ARMY(J,4,PN):B = ARM
        Y(J,5,PN): IF MAP(T,B,Ø)
         = J THEN MAP(T,B,Ø) = Ø
1E 1140 FOR K = J TO E: FOR L =
        Ø TO 6:ARMY(K,L,PN) = AR
        MY(K + 1,L,PN):ARMY(K +
        1,L,PN) = Ø:NEXTL
B9 1150 T = ARMY(K,4,PN):B = ARM
        Y(K,5,PN):MAP(T,B,Ø) = K
        : NEXT K
BA 1160 NX(PN) = NX(PN) - 1:J =
        E:SW = 1
1B 1170 NEXT J: IF SW THEN 1110
66 1180 FOR J = 1 TO E:ARMY(J,Ø,
        PN) = ARMY(J,Ø,PN) + ARM
        Y(J,2,PN)
83 1190 ARMY(J,2,PN) = ARMY(J,3,
        PN):ARMY(J,3,PN) = Ø
69 1200 ARMY(J,6,PN) = Ø
ED 1210 NEXT J:K = NX(1 - PN): F
        OR J = 1 TO K:ARMY(J,6,1
         - PN) = Ø: NEXT
47 1220 GOSUB 1400
B2 1230 IF BP > Ø THEN FOR J = Ø
         TO 1: FOR K = 1 TO BP:A
         = BTL(K,J,Ø):ARMY(A,6,J
        ) = ARMY(A,6,J) + 1: NEX
        T K,J
E5 1240 RETURN
53 1250 GOSUB 1400
13 1260 A = NX(Ø): IF (NX(1) > A
        ) THEN A = NX(1)
B8 1270 FOR J = Ø TO 1: FOR K =
        1 TO A:ARMY(K,6,J) = Ø:
        NEXT K,J
AB 1280 GOSUB 1230: RETURN
36 1290 FOR J = Ø TO 1:A = 1 - J
        :B = NX(J) - 1
B1 1300 FOR K = 1 TO B
BD 1310 IF ARMY(K,Ø,J) > = 1 THE
        N 1360
54 1320 FQ(2,A) = FQ(2,A) + ARMY
        (K,2,J) + ARMY(K,3,J): I
        F FQ(2,A) > 255 THEN C =
         FQ(2,A) - 255:FQ(3,A) =
         FQ(3,A) + C:FQ(2,A) = 2
        55
2E 1330 FQ(6,A) = FO(6,A) + ARMY
        (K,1,J): IF FQ(6,A) > 25
        5 THEN C = FQ(6,A) - 255
        :FQ(7,A) = FQ(7,A) + C:F
        Q(6,A) = 255
D2 1340 T = ARMY(K,4,J):B = ARMY
        (K,5,J): IF MAP(T,B,Ø) =
         K AND MAP(T,B,1) = J +
        1 THEN MAP(T,B,Ø) = Ø
CC 1350 FOR L = Ø TO 6:ARMY(K,L,
        J) = Ø: NEXT L
ED 1360 IF ARMY(K,6,J) > = 1 THE
        N 1390
BB 1370 FQ(4,J) = FQ(4,J) + ARMY
        (K,1,J):ARMY(K,1,J) = Ø
44 1380 IF FQ(4,J) > 255 THEN C
        = FQ(4,J) - 255:FQ(5,J)
        = FQ(5,J) + C:FQ(4,J) =
        255
6B 1390 NEXT K,J: RETURN
71 1400 BP = Ø
3D 1410 FOR J = Ø TO 8:J1 = (4 -
         J) * (4 - J > Ø):J2 = 8
         - (J > 4) * (J - 4)
1B 1420 FOR K = J1 TO J2:A = MAP
        (J,K,Ø):R = MAP(J,K,1)
BE 1430 IF (A = Ø) OR (R = Ø) TH
        EN 1480
AB 1440 IF ARMY(A,Ø,R - 1) < 1 T
        HEN 1480
D9 1450 T = J + 1:B = K: GOSUB 1
        490
19 1460 B = B - 1: GOSUB 1490
C1 1470 T = T - 1: GOSUB 1490
66 1480 NEXT K,J: RETURN
31 1490 IF T < Ø OR B < Ø OR T >
         8 OR B > 8 THEN RETURN
4B 1500 PA = MAP(T,B,Ø): IF PA =
         Ø THEN RETURN
1E 1510 IF MAP(T,B,1) = R THEN R
        ETURN
DE 1520 IF ARMY(PA,Ø,2 - R) < 1
        THEN RETURN
A2 1530 BP = BP + 1:BTL(BP,R - 1
        ,Ø) = A:BTL(BP,2 - R,Ø)
        = PA: RETURN
CC 1540 IF BP = Ø THEN RETURN
33 1550 FOR J = 1 TO BP
6D 1560 FOR K = Ø TO 1:A = 1 - K
21 1570 AN = BTL(J,K,Ø)
7D 1580 AS = ARMY(AN,Ø,K):HT = A
        RMY(AN,6,K):CT = INT (AS
         / HT) + 1
72 1590 BTL(J,A,1) = INT (CT * K
        A + 1)
EB 1600 BTL(J,A,2) = INT (CT * K
        B + 1)
75 1610 BTL(J,A,3) = INT (CT * K
        C + 1)
B6 1620 NEXT K,J
62 1630 FOR J = 1 TO BP:JØ = BTL
        (J,Ø,Ø):J1 = BTL(J,1,Ø)
B3 1640 GOSUB 1930
1E 1650 ARMY(JØ,Ø,Ø) = ARMY(JØ,Ø
        ,Ø) - A * BTL(J,Ø,1)
6F 1660 ARMY(J1,Ø,1) = ARMY(J1,Ø
        ,1) - B * BTL(J,1,1)
BF 1670 GOSUB 1930
A1 1680 C = A * BTL(J,Ø,2):ARMY(
        JØ,Ø,Ø) = ARMY(JØ,Ø,Ø) -
         C:ARMY(JØ,1,Ø) = ARMY(J
        Ø,1,Ø) + C
F2 1690 C = B * BTL(J,1,2):ARMY(
        J1,Ø,1) = ARMY(J1,Ø,1) -
         C:ARMY(J1,1,1) = ARMY(J
        1,1,1) + C
75 1700 GOSUB 1930
A9 1710 C = A * BTL(J,Ø,3):ARMY(
        JØ,Ø,Ø) = ARMY(JØ,Ø,Ø) -
         C:ARMY(JØ,3,Ø) = ARMY(J
        Ø,3,Ø) + C
FA 1720 C = B * BTL(J,1,3):ARMY(
        J1,Ø,1) = ARMY(J1,Ø,1) -
         C:ARMY(J1,3,1) = ARMY(J
        1,3,1) + C
86 1730 NEXT J
EF 1740 RETURN
F5 1750 A = 1 - PN:B = Ø
CB 1760 FOR J = Ø TO 8: FOR K =
        Ø TO 8
AA 1770 IF MAP(J,K,1) = PN + 1 T
        HEN B = B + 1
2B 1780 NEXT K,J
75 1790 FQ(1,PN) = FQ(1,PN) + B:
         IF FQ(1,PN) > 255 THEN
        B = FQ(1,PN) - 255:FQ(2,
        PN) = FQ(2,PN) + B:FQ(1,
        PN) = 255
B4 1800 T = 4:B = PN * 8
27 1810 IF MAP(T,B,Ø) < > Ø THEN
         RETURN
BC 1820 IF MAP(T,B,1) = PN + 1 T
        HEN FQ(Ø,A) = Ø:FQ(1,A)
        = Ø: RETURN
EC 1830 J = NX(A): IF J > 31 THE
        N RETURN
DE 1840 J1 = FQ(Ø,A): IF J1 < 1
        THEN 1900
6E 1850 NX(A) = NX(A) + 1
ØD 1860 MAP(T,B,Ø) = J:MAP(T,B,1
        ) = A + 1
4F 1870 ARMY(J,Ø,A) = J1
86 1880 FOR K = 1 TO 3:ARMY(J,K,
        A) = Ø: NEXT K
B4 1890 ARMY(J,4,A) = T:ARMY(J,5
        ,A) = B
42 1900 FOR K = Ø TO 19:FQ(K,A)
        = FQ(K + 1,A): NEXT K
EB 1910 FQ(20,A) = Ø
EB 1920 RETURN
FC 1930 A = Ø: FOR M = 1 TO 6: I
        F RND (1) < .5 THEN A =
        A + 1
56 1940 NEXT M:B = 6 - A: RETURN
83 1950 HOME : TEXT : VTAB 6: HT
        AB 10: PRINT "1> CAPTURE
         CAPITAL/FAR"
BB 1960 HTAB 10: PRINT "2> CAPTU
        RE CAPITAL/NEAR"
73 1970 HTAB 10: PRINT "3> OCCUP
        Y 8/12 CITIES"
7B 1980 HTAB 10: PRINT "4> CONTR
        OL 6/12 CITIES"
BB 1990 HTAB 10: PRINT "5> OCCUP
        Y 40/61 HEXES"
79 2000 VTAB 12: HTAB 10: PRINT
        "PRESS KEY TO SELECT GAM
        E"
```

```
84 2010 GET A$: IF A$ < "0" OR A
      $ > "5" THEN 2010
4B 2020 GN = VAL (A$): VTAB 5 +
      GN: HTAB 9: PRINT ">>";
8A 2030 VTAB 12: HTAB 4: PRINT "
      PREPARING GAME- ONE MOME
      NT PLEASE": RETURN
C7 2040 EA = 0: ON GN GOSUB 2110
      ,2130,2160,2170,2260
01 2050 IF EA = 0 THEN RETURN
09 2060 GOSUB 220: GOSUB 930
75 2070 TEXT : HOME
85 2080 HTAB 13: PRINT "PLAYER "
      EA" WINS"
4E 2090 HTAB 20 - ( LEN (EN$) -
      .5) / 2: PRINT EN$
8A 2100 HTAB 12: PRINT "(PRESS A
      NY KEY)": CALL 856: GET
      A$: RUN
A6 2110 IF MAP(CITY(2,0),CITY(2,
      1),1) = 1 THEN EA = 2:EN
      $ = "BLUE CAPTURED THE C
      APITAL": RETURN
68 2120 GOTO 2140
31 2130 IF MAP(CITY(3,0),CITY(3,
      1),1) = 1 THEN EA = 2:EN
      $ = "BLUE CAPTURED THE C
      APITAL": RETURN
CF 2140 IF MAP(CITY(1,0),CITY(1,
      1),1) = 2 THEN EA = 1:EN
      $ = "VIOLET CAPTURED THE
       CAPITAL"
E0 2150 RETURN
A7 2160 L = 8: GOTO 2180
AC 2170 L = 6
14 2180 C(1) = 0:C(2) = 0
11 2190 FOR J = 1 TO 12:T = CITY
      (J,0):8 = CITY(J,1)
EC 2200 R = MAP(T,8,1):C(R) = C(
      R) + 1
78 2210 IF GN = 4 THEN AN = MAP(
      T,8,0): IF R > 0 THEN IF
       AN = 0 OR ARMY(AN,6,R -
       1) > 0 THEN C(R) = C(R)
       - 1
79 2220 NEXT J
28 2230 IF C(1) = > L THEN EA =
      2:EN$ = "BLUE HAS CAPTUR
      ED " + STR$ (C(1)) + " C
      ITIES": RETURN
E9 2240 IF C(2) = > L THEN EA =
      1:EN$ = "VIOLET HAS CAPT
      URED " + STR$ (C(2)) + "
       CITIES"
EA 2250 RETURN
0E 2260 C(1) = 0:C(2) = 0
C6 2270 FOR J = 0 TO 8: FOR K =
      0 TO 8
75 2280 R = MAP(J,K,1):C(R) = C(
      R) + 1
18 2290 NEXT K,J
92 2300 IF C(1) = > 40 THEN EA =
       2:EN$ = "BLUE OCCUPIES
      " + STR$ (C(1)) + " HEXE
      S": RETURN
5D 2310 IF C(2) = > 40 THEN EA =
       1:EN$ = "VIOLET OCCUPIE
      S " + STR$ (C(2)) + " HE
      XES"
E0 2320 RETURN
C8 2330 IF PEEK (768) = 216 THEN
       RETURN
1F 2340 READ AD: IF AD = - 1 THE
      N RETURN
6D 2350 READ DT: IF DT = - 1 THE
      N 2340
3F 2360 POKE AD,DT:AD = AD + 1:
      GOTO 2350
2C 2370 DATA 8,4,0,4,8,0,8,4,0
      ,4,8
03 2380 DATA 5,5,3,3,6,3,2,5,5,2
      ,3,6
DD 2390 DATA 64,2,8,64,3,7,64,5,
      6,64,6,6
6A 2400 DATA 64,2,2,64,3,2,64,5,
      1,64,6,0
C1 2410 DATA 33024
5F 2420 DATA 0,0,0,0,0,0,0,0
47 2430 DATA -1,33048
3E 2440 DATA 131,140,176,192,192
      ,192,192,192
9C 2450 DATA 224,152,134,129,129
      ,129,129,129
19 2460 DATA 192,192,192,192,192
      ,176,140,131
C2 2470 DATA 129,129,129,129,129
      ,134,152,224
5C 2480 DATA -1,33088
DE 2490 DATA 42,42,0,0,0,0,0,0
17 2500 DATA 85,85,0,0,0,0,0,0
05 2510 DATA 0,0,0,0,0,0,170,170
1A 2520 DATA 0,0,0,0,0,0,213,213
03 2530 DATA 128,128,213,221,255
      ,255,221,213
43 2540 DATA 128,128,170,174,191
      ,191,174,170
3E 2550 DATA 85,93,127,127,93,85
      ,0,0
49 2560 DATA 42,46,63,63,46,42,0
      ,0
CC 2570 DATA 128,188,230,246,238
      ,230,188,128
04 2580 DATA 128,152,156,152,152
      ,152,188,128
F0 2590 DATA 128,188,230,176,140
      ,230,254,128
F0 2600 DATA 128,188,230,176,224
      ,230,188,128
F9 2610 DATA 128,176,184,180,254
      ,176,176,128
75 2620 DATA 128,254,134,190,224
      ,230,188,128
0C 2630 DATA 128,188,134,190,230
      ,230,188,128
A2 2640 DATA 128,254,224,176,152
      ,140,140,128
97 2650 DATA 128,188,230,188,230
      ,230,188,128
E0 2660 DATA 128,188,230,230,252
      ,176,152,128
96 2670 DATA -1,33248
33 2680 DATA 192,192,208,208,212
      ,212,213,213
D0 2690 DATA 128,128,130,130,138
      ,138,170,170
F3 2700 DATA 85,85,84,84,80,80,6
      4,64
C9 2710 DATA 42,42,10,10,2,2,0,0
0A 2720 DATA -1,33288
10 2730 DATA 128,252,230,230,254
      ,230,230,128
17 2740 DATA 128,190,230,230,190
      ,230,254,128
E8 2750 DATA 128,188,230,134,134
      ,230,190,128
06 2760 DATA 128,190,230,230,230
      ,230,190,128
A4 2770 DATA 128,254,134,134,190
      ,134,254,128
A3 2780 DATA 128,254,134,134,190
      ,134,134,128
E0 2790 DATA -1,33400
E5 2800 DATA 128,128,128,128,224
      ,224,224,252
51 2810 DATA 128,128,128,128,129
      ,129,129,143
0C 2820 DATA 252,224,224,224,128
      ,128,128,128
06 2830 DATA 143,129,129,129,128
      ,128,128,128
C5 2840 DATA 128,128,144,144,144
      ,149,149,128
04 2850 DATA 128,128,130,130,130
      ,170,170,128
19 2860 DATA 128,149,149,144,144
      ,144,128,128
C8 2870 DATA 128,170,170,130,130
      ,130,128,128
0E 2880 DATA 0,0,16,16,16,21,21,
      0
AA 2890 DATA 0,0,2,2,2,42,42,0
8D 2900 DATA 0,21,21,16,16,16,0,
      0
14 2910 DATA 0,42,42,2,2,2,0,0
6D 2920 DATA -1,33528
99 2930 DATA 0,0,0,0,127,0,0,0
1D 2940 DATA -1,768
48 2950 DATA 216,120,133,69,134,
      70,132,71
7D 2960 DATA 166,7,10,10,176,4,1
      6,62
04 2970 DATA 48,4,16,1,232,232,1
      0,134
20 2980 DATA 27,24,101,6,133,26,
      144,2
1A 2990 DATA 230,27,165,40,133,8
      ,165,41
D7 3000 DATA 41,3,5,230,133,9,16
      2,8
40 3010 DATA 160,0,177,26,36,50,
      48,2
59 3020 DATA 73,127,164,36,145,8
      ,230,26
07 3030 DATA 208,2,230,27,165,9,
      24,105
52 3040 DATA 4,133,9,202,208,226
      ,165,69
5A 3050 DATA 166,70,164,71,88,76
      ,240,253
09 3060 DATA 169,120,133,254,169
      ,64,133,255
A9 3070 DATA 169,15,160,7,145,25
      4,136,16
59 3080 DATA 251,44,85,192,44,80
      ,192,169
30 3090 DATA 15,205,176,192,208,
      251,44,84
5C 3100 DATA 192,44,81,192,160,2
      51,162,72
30 3110 DATA 232,208,253,200,208
      ,250,44,0
D1 3120 DATA 192,16,222,96
52 3130 DATA -1,-1
```

Program 6. Hex War For IBM PC/PCjr

Version by Patrick Parrish, Programming Supervisor

```
HK 0 KEY OFF:WIDTH 40:DEF SEG=0:
     POKE 1047,PEEK(1047) OR 64:
     SCREEN 1:COLOR 0,1:CLS:LOCA
     TE 12,15,0:PRINT "PLEASE WA
     IT":GOTO 20
AG 2 DEF SEG=0:POKE 1050,PEEK(10
     52)
FP 3 QV=0:A$=RIGHT$(INKEY$,1):IF
      LEN(A$)=0 THEN 3
IP 4 IF A$=CHR$(77) THEN QV=3:GO
     TO 8
NK 5 IF A$=CHR$(75) THEN QV=7:GO
     TO 8
AC 6 IF A$=CHR$(72) THEN QV=1:GO
     TO 8
OF 7 IF A$=CHR$(80) THEN QV=5
HP 8 J=QV-128*(A$=" "):RETURN
CF 20 GOSUB 3710
CI 40 DIM ARMY(31,6,1),BTL(64,1,
      3),MAP(9,9,2),FQ(20,1),NX(
      1),C(2)
E0 50 CN=12:DIM CIT(CN,1)
HP 60 RANODMIZE(TIMER)
EJ 70 PN=1:ME=31
JC 80 MM=3:REM MAX MOVES
FF 90 KA=1/48:KB=1/48:KC=1/32
KG 260 RESTORE 270:FOR J=1 TO CN
       :FOR K=0 TO 1:READ CIT(J,
```

```
         K):NEXT K:MAP(CIT(J,Ø),CI
         T(J,1),2)=1:NEXT J
DC 270 DATA 8,4,Ø,4,8,Ø,Ø,8,4,Ø,
         4,8
DM 280 DATA 5,5,3,3,6,3,2,5,5,2,
         3,6
LK 300 GOSUB 600:GOSUB 3200:GOSU
         B 600
JH 310 LOCATE 24,17:PRINT "HEX W
         AR";
DI 330 LOCATE 1,11:PRINT "PLEASE
          WAIT A MOMENT"
BD 340 GOSUB 1500
IH 410 REM CLEAR KEYBD
CH 420 GOSUB 1900:GOSUB 600:GOSU
         B 1710:REM FIND BATTLES
DD 430 IF PN=1 THEN PUT (280,160
         ),S5,PSET ELSE PUT (280,1
         60),S6,PSET
FD 431 GOSUB 800:LOCATE 21,36:PR
         INT"  ":REM KEYBD
CL 440 COLOR 14:GOSUB 2100:REM B
         ATTLES AGAIN
ML 450 COLOR 1:GOSUB 2600:REM RE
         SOLVE
HI 460 COLOR 1:GOSUB 2100:REM PO
         ST-BATTLE
GG 470 COLOR 2:GOSUB 2200:REM SP
         LIT PRISONERS
EM 480 GOSUB 2900:REM REINFORCEM
         ENTS
EM 490 COLOR 6:GOSUB 3400
OK 500 PN=1-PN:FT=Ø:PP=Ø
DG 510 GOTO 420
PF 600 CLS:COLOR Ø
KJ 610 FOR R=11 TO 1 STEP -2:FOR
          C=12-R TO R+26 STEP 4:PU
         T (C*8,R*8),S1Ø:NEXT C,R
FM 620 FOR R=13 TO 21 STEP 2:FOR
          C=R-1Ø TO 49-R STEP 4:PU
         T (C*8,R*8),S1Ø:NEXT C,R
BM 630 FOR R=12 TO 1 STEP -2:FOR
          C=14-R TO R+28 STEP R+28
         -(14-R)-1:LOCATE R,C:PRIN
         T " ":NEXT C,R
DI 640 FOR R=13 TO 21 STEP 2:FOR
          C=R-11 TO 53-R STEP 53-R
         -(R-11)-1:LOCATE R,C:PRIN
         T " ":NEXT C,R
EH 650 FOR I=2 TO 23 STEP 21:LOC
         ATE I,12,Ø:PRINT "
                   ":NEXT
DO 670 FOR J=1 TO 12:GOSUB 710:N
         EXT
BP 680 J=1:IF GN=1 THEN GOSUB 71
         8:J=2:GOSUB 715
AH 690 IF GN=2 THEN GOSUB 718:J=
         3:GOSUB 715
DG 700 LOCATE 1,1:RETURN
MG 710 K=CIT(J,Ø):L=CIT(J,1):X=(
         K-L)*2+19:Y=(12-(K+L))*2+
         3:PUT (X*8+3,Y*8+3),S2,PS
         ET:RETURN
KJ 715 K=CIT(J,Ø):L=CIT(J,1):X=(
         K-L)*2+19:Y=(12-(K+L))*2+
         3:PUT (X*8,Y*8),S3,PSET:R
         ETURN
PK 718 K=CIT(J,Ø):L=CIT(J,1):X=(
         K-L)*2+19:Y=(12-(K+L))*2+
         3:PUT (X*8,Y*8),S4,PSET:R
         ETURN
JJ 800 IF NX(PN)<2 THEN RETURN
LP 805 HT=4:HB=4:GOSUB 1000
DI 810 MV=Ø:CT=Ø:CB=Ø:PK=Ø:K=Ø
IB 820 FOR J=1 TO NX(PN)-1:IF AR
         MY(J,Ø,PN)>Ø AND ARMY(J,6
         ,PN)<1 THEN K=1:J=NX(PN)-
         1
JD 830 NEXT J:IF K=Ø THEN RETURN
CD 840 IF A$=CHR$(27) THEN A$=""
         :RETURN
AH 850 GOSUB 2:IF J=Ø THEN 840 E
         LSE IF (J AND 128) THEN 1
         100 ELSE IF (J AND 1)=Ø T
         HEN 840
BL 860 J=(J-1)/2:IF J AND 1 THEN
          B1=HB+J-2:T1=HT ELSE T1=
         HT+1-J:B1=HB
PP 870 IF T1<Ø OR T1>8 THEN 840
EB 880 IF B1<Ø OR B1>8 THEN 840
FE 890 S1=T1+B1:IF S1<4 OR S1>12
          THEN 840
BA 895 HB=B1:HT=T1:GOSUB 1000
DB 896 LOCATE 1,1:FOR Z=1 TO 6:P
         RINT"              ":NEXT
CJ 900 QN=MAP(HT,HB,Ø):IF QN=Ø T
         HEN LOCATE 1,1:GOTO 840 E
         LSE Q1=MAP(HT,HB,1)-1
GJ 910 LOCATE 1,1:PRINT USING "#
         ###";QN:PRINT"------"
CK 920 FOR J=Ø TO 3:PRINT USING
         "####";ARMY(QN,J,Q1):NEXT
DI 930 LOCATE 1,1:GOTO 840
ML 1000 SX=150+16*(HT-HB):SY=214
          -16*(HT+HB)
CC 1005 IF MAP(HT,HB,2)=1 THEN P
          UT (OX,OY),S8:PUT (SX,SY
          ),S7:OX=SX:OY=SY:PP=1:RE
          TURN
AF 1007 IF PP THEN PUT (OX,OY),S
          7:PUT (SX,SY),S8:OX=SX:O
          Y=SY:PP=Ø:RETURN
DJ 1010 IF FT THEN PUT (OX,OY),S
          8:PUT (SX,SY),S8:OX=SX:O
          Y=SY:RETURN
MJ 1015 PUT (SX,SY),S8:OX=SX:OY=
          SY:FT=1
IE 1030 RETURN
JF 1100 IF PK=1 THEN 1200
BA 1120 IF MAP(HT,HB,1)<>PN+1 OR
           MAP(HT,HB,Ø)=Ø THEN 810
KG 1130 AN=MAP(HT,HB,Ø):IF ARMY(
          AN,6,PN)<>Ø THEN 810
PJ 1140 PK=1:CT=HT:CB=HB:CS=ARMY
          (AN,Ø,PN)
MO 1150 SOUND 2200,1Ø
FH 1160 GOTO 840
KK 1200 J=(HT=CT) AND (HB=CB)
ED 1210 IF J AND MV=Ø THEN 810
HM 1220 IF J AND MV>Ø THEN 1420
NH 1230 AS=ARMY(MAP(HT,HB,Ø),Ø,P
          N):IF (AS>ME AND CS>ME)
          OR (MAP(HT,HB,1)-1=1-PN
          AND AS>Ø) THEN 840
DE 1240 OT=ABS(CT-HT):OB=ABS(CB-
          HB):TL=OB+OT:IF NOT (TL=
          1 OR (CT+CB=HT+HB AND OT
          =1)) THEN 840
CF 1250 MG=MAP(HT,HB,Ø):IF MG=Ø
          THEN 1300
CL 1260 FOR J=Ø TO 3:ARMY(MG,J,P
          N)=ARMY(MG,J,PN)+ARMY(AN
          ,J,PN):ARMY(AN,J,PN)=Ø:N
          EXT J
DM 1270 ARMY(MG,6,PN)=1:MAP(CT,C
          B,Ø)=Ø
BN 1280 CS=ARMY(MG,Ø,PN):AN=MG:M
          V=MM+1
BB 1290 GOTO 1380
JF 1300 NB=MAP(HT,HB,1)-1:MV=MV+
          1:IF NB<>PN THEN MV=MV+1
IL 1310 MAP(CT,CB,Ø)=Ø
IM 1320 MAP(HT,HB,Ø)=AN:MAP(HT,H
          B,1)=PN+1:ARMY(AN,4,PN)=
          HT:ARMY(AN,5,PN)=HB:IF M
          V>=MM THEN ARMY(AN,6,PN)
          =1
CJ 1330 K=Ø:FOR J=-1 TO 1 STEP 2
          :J1=HT+J:J2=HB+J:J3=HB-J
          :IF J1<Ø OR J1>8 THEN 13
          4Ø ELSE IF MAP(J1,HB,Ø)>
          Ø THEN IF MAP(J1,HB,1)=2
          -PN THEN K=1:J=1:GOTO 13
          6Ø
DQ 1340 IF J2<Ø OR J2>8 THEN 135
          Ø ELSE IF MAP(HT,J2,Ø)>Ø
           THEN IF MAP(HT,J2,1)=2-
          PN THEN K=1:J=1:GOTO 136
          Ø
FB 1350 IF J3<Ø OR J3>8 OR J1<Ø
          OR J1>8 THEN 1360 ELSE I
          F MAP(J1,J3,Ø)>Ø THEN IF
           MAP(J1,J3,1)=2-PN THEN
          K=1:J=1
HC 1360 NEXT J
NM 1370 IF K=1 THEN ARMY(AN,6,PN
          )=1:MV=MM+1
DL 1380 A=PN:J=CT:K=CB:C=Ø:D=Ø:G
          OSUB 1830
MF 1390 J=HT:K=HB:C=CS:D=ARMY(AN
          ,6,PN):GOSUB 1830
MJ 1400 CT=HT:CB=HB
LH 1410 IF MV<MM THEN 840
PJ 1420 ARMY(AN,6,PN)=1:J=HT:K=H
          B:C=CS:D=1:GOSUB 1830
DA 1430 GOTO 810
BI 1500 RESTORE 1540:FOR J=Ø TO
          1:NX(J)=5:FOR K=1 TO 4:R
          EAD A,B,C
DA 1510 ARMY(K,Ø,J)=A:ARMY(K,4,J
          )=B:ARMY(K,5,J)=C:MAP(B,
          C,Ø)=K:MAP(B,C,1)=J+1
EH 1520 NEXT K,J
OK 1530 REM STRENGTH, T-POS, B-P
          OS
EL 1540 DATA 64,2,8,64,3,7,64,5,
          6,64,6,6:REM BLUE
HD 1550 DATA 64,2,2,64,3,2,64,5,
          1,64,6,Ø:REM VIOLET
EG 1600 REM SET RANDOM REINFORCE
          MENTS
AH 1610 FOR J=Ø TO 1:FOR K=Ø TO
          2Ø
PH 1620 A=INT(RND(1)*K*3):FOR L=
          1 TO 5:A=A+INT(RND(1)*21
          -8):NEXT L:IF A<16 THEN
          A=Ø ELSE A=(A+K*8) AND 2
          54
DJ 1630 FQ(K,J)=A:NEXT K,J
JD 1640 RETURN
FF 1700 REM ARMIES->MAP
BE 1710 FOR J=Ø TO 8:FOR K=Ø TO
          8
AH 1720 A=MAP(J,K,1):IF A THEN A
          =A-1:GOSUB 1800
FD 1730 NEXT K,J
LI 1740 FOR A=Ø TO 1:E=13+A*12:F
          =A*22:DX=2-4*A:D=Ø
AA 1750 FOR J=Ø TO 8:C=FQ(J,A):G
          OSUB 1840
DM 1760 E=E+DX*2:IF J>3 THEN F=F
          +DX:E=E-DX
OK 1770 NEXT J,A
KB 1780 RETURN
HM 1800 B=MAP(J,K,Ø)
PF 1810 C=ARMY(B,Ø,A)
BI 1820 D=ARMY(B,6,A)
DF 1830 E=(J-K+10)*2-1:F=(13-J-K
          )*2+1:REM T&B TO X/Y
HM 1840 IF A THEN PUT (E*8,F*8),
          S5,PSET:LOCATE F+A+1,E+1
          :PRINT "  "; ELSE PUT (E
          *8,(F+1)*8),S6,PSET:LOCA
          TE F+A+1,E+1:PRINT "  ";
EB 1850 IF C=Ø THEN RETURN
LD 1870 LOCATE F+A+1,E+1:PRINT R
          IGHT$("Ø"+HEX$(C),2);
BP 1880 IF D AND A=Ø THEN LOCATE
           F+2,E+1:PRINT "  ";:PUT
           (E*8+1,(F+1)*8+1),S11 E
          LSE IF D AND A=1 THEN LO
          CATE F+1,E+1:PRINT "  ";
          :PUT (E*8+1,F*8+1),S9
KB 1890 RETURN
HJ 1900 SW=Ø:E=NX(PN)-1:IF E<1 T
          HEN RETURN
JN 1910 FOR J=1 TO E-1:IF ARMY(J
          ,Ø,PN)>=1 THEN 1970
HC 1930 FOR K=J TO E:FOR L=Ø TO
```

```
        6:ARMY(K,L,PN)=ARMY(K+1,L,PN):ARMY(K+1,L,PN)=Ø:NEXT L
CB 1940 T=ARMY(K,4,PN):B=ARMY(K,5,PN):MAP(T,B,Ø)=K
IG 1950 NEXT K
IE 1960 NX(PN)=NX(PN)-1:J=E:SW=1
BL 1970 NEXT J:IF SW THEN 1900
BP 2000 FOR J=1 TO E:ARMY(J,Ø,PN)=ARMY(J,Ø,PN)+ARMY(J,2,PN)
MI 2010 ARMY(J,2,PN)=ARMY(J,3,PN):ARMY(J,3,PN)=Ø
PM 2020 ARMY(J,6,PN)=Ø
GF 2030 NEXT J:K=NX(1-PN):FOR J=1 TO K:ARMY(J,6,1-PN)=Ø:NEXT
FG 2040 GOSUB 2400
IL 2050 IF BP>Ø THEN FOR J=Ø TO 1:FOR K=1 TO BP:A=BTL(K,J,Ø):ARMY(A,6,J)=ARMY(A,6,J)+1:NEXT K,J
JD 2060 RETURN
FM 2100 GOSUB 2400
BL 2110 A=NX(Ø):IF NX(1)>A THEN A=NX(1)
EM 2120 FOR J=Ø TO 1:FOR K=1 TO A:ARMY(K,6,J)=Ø:NEXT K,J
GH 2130 GOSUB 2050
JK 2140 RETURN
CO 2200 FOR J=Ø TO 1:A=1-J:B=NX(J)-1
CK 2210 FOR K=1 TO B
DG 2220 IF ARMY(K,Ø,J)>=1 THEN 2280
PM 2230 FQ(2,A)=FQ(2,A)+ARMY(K,2,J)+ARMY(K,3,J):IF FQ(2,A)>255 THEN C=FQ(2,A)-255:FQ(3,A)=FQ(3,A)+C:FQ(2,A)=255
EP 2240 FQ(6,A)=FQ(6,A)+ARMY(K,1,J):IF FQ(6,A)>255 THEN C=FQ(6,A)-255:FQ(7,A)=FQ(7,A)+C:FQ(6,A)=255
BB 2250 IF MAP(ARMY(K,4,J),ARMY(K,5,J),Ø)=K AND MAP(ARMY(K,4,J),ARMY(K,5,J),1)=J+1 THEN MAP(ARMY(K,4,J),ARMY(K,5,J),Ø)=Ø
NE 2260 FOR L=Ø TO 6:ARMY(K,L,J)=Ø:NEXT L
KP 2280 IF ARMY(K,6,J)>=1 THEN 2320:REM EVACUATE INJURED
PI 2290 FQ(4,J)=FQ(4,J)+ARMY(K,1,J):ARMY(K,1,J)=Ø
MO 2300 IF FQ(4,J)>255 THEN C=FQ(4,J)-255:FQ(5,J)=FQ(5,J)+C:FQ(4,J)=255
FK 2320 NEXT K,J:RETURN
OO 2400 BP=Ø
IO 2410 FOR J=Ø TO 8:J1=(J-4)*(4-J>Ø):J2=8-(J>4)*(4-J):FOR K=J1 TO J2
BF 2420 A=MAP(J,K,Ø)
OO 2430 R=MAP(J,K,1)
JC 2440 IF A=Ø OR R=Ø THEN 2490
JN 2450 IF ARMY(A,Ø,R-1)<1 THEN 2490
AB 2460 T=J+1:B=K:GOSUB 2500
CC 2470 B=B-1:GOSUB 2500
CB 2480 T=T-1:GOSUB 2500
GB 2490 NEXT K,J:RETURN
OM 2500 IF T<Ø OR B<Ø OR T>8 OR B>8 THEN RETURN
IA 2510 PA=MAP(T,B,Ø):IF PA=Ø THEN RETURN
CM 2520 IF MAP(T,B,1)=R THEN RETURN
CO 2530 IF ARMY(PA,Ø,2-R)<1 THEN RETURN
KF 2540 BP=BP+1:BTL(BP,R-1,Ø)=A:BTL(BP,2-R,Ø)=PA:RETURN
AH 2600 IF BP=Ø THEN RETURN
MI 2610 FOR J=1 TO BP
ED 2620 FOR K=Ø TO 1:A=1-K
BH 2630 AN=BTL(J,K,Ø)
BH 2640 AS=ARMY(AN,Ø,K):HT=ARMY(AN,6,K):CT=INT(AS/HT)+1
MG 2650 BTL(J,A,1)=INT(CT*KA+1)
PA 2660 BTL(J,A,2)=INT(CT*KB+1)
CK 2670 BTL(J,A,3)=INT(CT*KC+1)
GM 2680 NEXT K,J
IB 2700 FOR J=1 TO BP:JØ=BTL(J,Ø,Ø):J1=BTL(J,1,Ø)
EA 2710 GOSUB 3100
DC 2720 ARMY(JØ,Ø,Ø)=ARMY(JØ,Ø,Ø)-A*BTL(J,Ø,1)
NP 2730 ARMY(J1,Ø,1)=ARMY(J1,Ø,1)-B*BTL(J,1,1)
FJ 2740 GOSUB 3100
CI 2750 C=A*BTL(J,Ø,2):ARMY(JØ,Ø,Ø)=ARMY(JØ,Ø,Ø)-C:ARMY(JØ,1,Ø)=ARMY(JØ,1,Ø)+C
MP 2760 C=B*BTL(J,1,2):ARMY(J1,Ø,1)=ARMY(J1,Ø,1)-C:ARMY(J1,1,1)=ARMY(J1,1,1)+C
FC 2770 GOSUB 3100
DB 2780 C=A*BTL(J,Ø,3):ARMY(JØ,Ø,Ø)=ARMY(JØ,Ø,Ø)-C:ARMY(JØ,3,Ø)=ARMY(JØ,3,Ø)+C
OI 2790 C=B*BTL(J,1,3):ARMY(J1,Ø,1)=ARMY(J1,Ø,1)-C:ARMY(J1,3,1)=ARMY(J1,3,1)+C
HL 2800 NEXT J
JP 2810 RETURN
JE 2900 A=1-PN:B=Ø
MJ 2910 FOR J=Ø TO 8:FOR K=Ø TO 8
OA 2920 IF MAP(J,K,1)=PN+1 THEN B=B+1
FD 2930 NEXT K,J
MD 2950 FQ(1,PN)=FQ(1,PN)+B
CM 2955 IF FQ(1,PN)>255 THEN B=FQ(1,PN)-255:FQ(2,PN)=FQ(2,PN)+B:FQ(1,PN)=255
HA 2960 T=4:B=PN*8
MP 2970 IF MAP(T,B,Ø)<>Ø THEN RETURN
MM 2980 IF MAP(T,B,1)=PN+1 THEN FQ(Ø,A)=Ø:FQ(1,A)=Ø:GOTO 3060
NC 2990 J=NX(A):IF J>31 THEN RETURN
BF 3000 J1=FQ(Ø,A):IF J1<1 THEN 3060
IA 3010 NX(A)=NX(A)+1
EP 3020 MAP(T,B,Ø)=J:MAP(T,B,1)=A+1
MC 3030 ARMY(J,Ø,A)=J1
EG 3040 FOR K=1 TO 3:ARMY(J,K,A)=Ø:NEXT K
HM 3050 ARMY(J,4,A)=T:ARMY(J,5,A)=B
PF 3060 FOR K=Ø TO 19:FQ(K,A)=FQ(K+1,A):NEXT K
IA 3070 FQ(20,A)=Ø
JF 3080 RETURN
IC 3100 A=Ø:FOR M=1 TO 6:IF RND(1)<.5 THEN A=A+1
KO 3110 NEXT M:B=6-A:RETURN
PO 3200 REM WINDOW
FM 3210 LOCATE 8,7:PRINT STRING$(9,19)" SCENARIO "STRING$(9,19):LOCATE 9,7:GOSUB 3310
PB 3220 LOCATE 10,7:PRINT CHR$(19)" 1> CAPTURE CAPITAL/FAR  "CHR$(19)
PN 3230 LOCATE 11,7:PRINT CHR$(19)" 2> CAPTURE CAPITAL/NEAR "CHR$(19)
NG 3240 LOCATE 12,7:PRINT CHR$(19)" 3> OCCUPY  8/12 CITIES  "CHR$(19)
FA 3250 LOCATE 13,7:PRINT CHR$(19)" 4> CONTROL  6/12 CITIES  "CHR$(19)
LF 3260 LOCATE 14,7:PRINT CHR$(19)" 5> OCCUPY  40/61 HEXES   "CHR$(19):LOCATE 15,7:GOSUB 3310
IO 3270 LOCATE 16,7:PRINT STRING$(28,19)
OI 3280 A$=INKEY$:IF A$="" THEN 3280 ELSE GN=VAL(A$):IF GN<1 OR GN>5 THEN 3280
BH 3290 LOCATE 9+GN,9:PRINT CHR$(16):FOR TO=1 TO 1000:NEXT
ID 3300 RETURN
HO 3310 PRINT CHR$(19)SPC(26)CHR$(19):RETURN
IA 3400 A=Ø:ON GN GOSUB 3430,3450,3480,3490,3580
EK 3410 IF A=Ø THEN RETURN ELSE EN$=C$:EA=A:GOSUB 600:GOSUB 1710:A=EA
ML 3420 LOCATE 1,1:PRINT "PLAYER "A"WINS  ":PRINT EN$:PRINT "(PRESS ANY KEY) "
CL 3425 A$=INKEY$:IF A$="" THEN 3425 ELSE RUN
EH 3430 IF MAP(CIT(2,Ø),CIT(2,1),1)=1 THEN A=2:C$="BLUE CAPTURED THE CAPITAL ":RETURN
BA 3440 GOTO 3460
II 3450 IF MAP(CIT(3,Ø),CIT(3,1),1)=1 THEN A=2:C$="BLUE CAPTURED THE CAPITAL ":RETURN
PF 3460 IF MAP(CIT(1,Ø),CIT(1,1),1)=2 THEN A=1:C$="VIOLET CAPTURED THE CAPITAL "
KK 3470 RETURN
KL 3480 L=8:GOTO 3500
MJ 3490 L=6
OM 3500 C(1)=Ø:C(2)=Ø
ND 3510 FOR J=1 TO 12:T=CIT(J,Ø):B=CIT(J,1)
OI 3520 R=MAP(T,B,1):C(R)=C(R)+1
KB 3530 IF GN=4 THEN AN=MAP(T,B,Ø):IF R>Ø THEN IF AN=Ø OR ARMY(AN,6,R-1)>Ø THEN C(R)=C(R)-1
MC 3540 NEXT J
ME 3550 IF C(1)=>L THEN A=2:C$="BLUE HAS CAPTURED"+STR$(C(1))+" CITIES ":RETURN
NO 3560 IF C(2)=>L THEN A=1:C$="VIOLET HAS CAPTURED"+STR$(C(2))+" CITIES "
KM 3570 RETURN
PE 3580 C(1)=Ø:C(2)=Ø
IK 3590 FOR J=Ø TO 8:FOR K=Ø TO 8
OK 3600 R=MAP(J,K,1):C(R)=C(R)+1
FI 3610 NEXT K,J
KM 3620 IF C(1)=>40 THEN A=2:C$="BLUE OCCUPIES"+STR$(C(1))+" HEXES":RETURN
CA 3630 IF C(2)=>40 THEN A=1:C$="VIOLET OCCUPIES"+STR$(C(2))+" HEXES"
JF 3640 RETURN
EE 3700 REM DEFINE SHAPES
OP 3710 DEFINT S
IB 3720 RESTORE 3840:READ X,Y:E=(4+INT((X+7)/8)*Y)/2:DIM S10(E):S10(Ø)=X:S10(1)=Y:FOR I=2 TO E:READ S10(I):NEXT
PJ 3730 READ X,Y:E=(4+INT((X+7)/8)*Y)/2:DIM S2(E):S2(Ø)=X:S2(1)=Y:FOR I=2 TO E:READ S2(I):NEXT
ME 3740 READ X,Y:E=(4+INT((X+7)/8)*Y)/2:DIM S3(E):S3(Ø)=
```

```
          X:S3(1)=Y:FDR I=2 TD E:R
          EAD S3(I):NEXT
MP 3750 READ X,Y:E=(4+INT((X+7)/
          8)*Y)/2:DIM S4(E):S4(Ø)=
          X:S4(1)=Y:FDR I=2 TO E:R
          EAD S4(I):NEXT
LK 3760 READ X,Y:E=(4+INT((X+7)/
          8)*Y)/2:DIM S5(E):S5(Ø)=
          X:S5(1)=Y:FOR I=2 TD E:R
          EAD S5(I):NEXT
JF 3770 READ X,Y:E=(4+INT((X+7)/
          8)*Y)/2:DIM S6(E):S6(Ø)=
          X:S6(1)=Y:FDR I=2 TD E:R
          EAD S6(I):NEXT
IA 3780 READ X,Y:E=(4+INT((X+7)/
          8)*Y)/2:DIM S7(E):S7(Ø)=
          X:S7(1)=Y:FDR I=2 TO E:R
          EAD S7(I):NEXT
HL 3790 READ X,Y:E=(4+INT((X+7)/
          8)*Y)/2:DIM S8(E):S8(Ø)=
          X:S8(1)=Y:FDR I=2 TD E:R
          EAD S8(I):NEXT
EK 3800 READ X,Y:E=(4+INT((X+7)/
          8)*Y)/2:DIM S9(E):S9(Ø)=
          X:S9(1)=Y:FDR I=2 TD E:R
          EAD S9(I):NEXT
DA 3810 READ X,Y:E=(4+INT((X+7)/
          8)*Y)/2:DIM S11(E):S11(Ø
          )=X:S11(1)=Y:FDR I=2 TD
          E:READ S11(I):NEXT
JD 3820 RETURN
HH 3830 REM HEX SHAPE S1Ø
CH 3840 DATA &H2Ø,&H1Ø,&HFC,&H3F
          ØØ,&HFC,&H3FØØ,&HFFØ3,&H
          CØFF
BE 3850 DATA &HFFØØ,&HFF,&HFØØ,&
          HFØ,&HFØØ,&HFØ,&HFØØ,&HF
          Ø
BF 3860 DATA &HFØØ,&HFØ,&HFØØ,&H
          FØ,&HFØØ,&HFØ,&HFØØ,&HFØ
FB 3870 DATA &HFØØ,&HFØ,&HFFØØ,&
          HFF,&HFFØ3,&HCØFF,&HFC,&
          H3FØØ
ED 3880 DATA &HFC,&H3FØØ,&HØ
AD 3890 REM CITY SHAPE S2
PL 3900 DATA &H14,&HA,&HAAØA,&H2
          AØØ,&H8ØAA,&HAØ,&HAØAØ,&
          HAØØØ
DB 3910 DATA &HAØ,&HAØAØ,&HAØØØ,
          &HAØ,&HAØAØ,&HAØØØ,&HAA2
          A,&HA8Ø
CM 3920 DATA &HAA,&HØ
HC 3930 REM CAP.PUR SHAPE S3
IN 3940 DATA &H2Ø,&H1Ø,&HAAØØ,&H
          AA,&HAAØØ,&HAA,&HAAØØ,&H
          AA
DI 3950 DATA &HAAØØ,&HAA,&HAAAA,
          &HAAAA,&HAAAA,&HAAAA,&HA
          AAA,&HAAAA
BP 3960 DATA &HAAAA,&HAAAA,&HAAA
          A,&HAAAA,&HAAAA,&HAAAA,&
          HAAAA,&HAAAA
EO 3970 DATA &HAAAA,&HAAAA,&HAAØ
          Ø,&HAA,&HAAØØ,&HAA,&HAAØ
          Ø,&HAA
ON 3980 DATA &HAAØØ,&HAA,&HØ
IC 3990 REM CAP.BLU SHAPE S4
HA 4000 DATA &H2Ø,&H1Ø,&H55ØØ,&H
          55,&H55ØØ,&H55,&H55ØØ,&H
          55
BL 4010 DATA &H55ØØ,&H55,&H5555,
          &H5555,&H5555,&H5555,&H5
          555,&H5555
HC 4020 DATA &H5555,&H5555,&H555
          5,&H5555,&H5555,&H5555,&
          H5555,&H5555
DB 4030 DATA &H5555,&H5555,&H55Ø
          Ø,&H55,&H55ØØ,&H55,&H55Ø
          Ø,&H55
JA 4040 DATA &H55ØØ,&H55,&HØ
DN 4050 REM TOP.PUR SHAPE S5
CM 4060 DATA &H2Ø,&H8,&H2ØØ,&H8Ø
          ,&HAØØ,&HAØ,&H2AØØ,&HA8
CH 4070 DATA &HAAØØ,&HAA,&HAAØ2,
          &H8ØAA,&HAAØA,&HAØAA,&HA
          A2A,&HA8AA
BC 4080 DATA &HØ,&HØ,&HØ
DH 4090 REM TOP.BLU SHAPE S6
FF 4100 DATA &H2Ø,&H8,&H5515,&H5
          455,&H55Ø5,&H5Ø55,&H55Ø1
          ,&H4Ø55
DE 4110 DATA &H55ØØ,&H55,&H15ØØ,
          &H54,&H5ØØ,&H5Ø,&H1ØØ,&H
          4Ø
FC 4120 DATA &HØ,&HØ,&HØ
DC 4130 REM CURSOR.PUR SHAPE S7
EK 4140 DATA &H28,&H14,&H2AØØ,&H
          A8AA,&HØ,&HAA2A,&HA8,&HØ
CN 4150 DATA &HØ,&HØ,&HØ,&HØ,&HA
          Ø,&HØ,&HAØØA,&HØ
NJ 4160 DATA &HAØØ,&HAØ,&HØ,&HAØ
          ØA,&HØ,&HAØØ,&HAØ,&HØ
DL 4170 DATA &HAØØA,&HØ,&HAØØ,&H
          AØ,&HØ,&HAØØA,&HØ,&HAØØ
NB 4180 DATA &HAØ,&HØ,&HAØØA,&HØ
          ,&HAØØ,&HAØ,&HØ,&HAØØA
EN 4190 DATA &HØ,&HAØØ,&HØ,&HØ,&
          HØ,&HØ,&HØ,&H2AØØ
HD 4200 DATA &HA8AA,&HØ,&HAA2A,&
          HA8,&HØ
PD 4210 REM CURSOR.BLU SHAPE S8
PM 4220 DATA &H28,&H14,&H15ØØ,&H
          5455,&HØ,&H5515,&H54,&HØ
BJ 4230 DATA &HØ,&HØ,&HØ,&HØ,&H5
          Ø,&HØ,&H5ØØ5,&HØ
JB 4240 DATA &H5ØØ,&H5Ø,&HØ,&H5Ø
          Ø5,&HØ,&H5ØØ,&H5Ø,&HØ
MP 4250 DATA &H5ØØ5,&HØ,&H5ØØ,&H
          5Ø,&HØ,&H5ØØ5,&HØ,&H5ØØ
QJ 4260 DATA &H5Ø,&HØ,&H5ØØ5,&HØ
          ,&H5ØØ,&H5Ø,&HØ,&H5ØØ5
DB 4270 DATA &HØ,&H5ØØ,&HØ,&HØ,&
          HØ,&HØ,&HØ,&H15ØØ
KF 4280 DATA &H5455,&HØ,&H5515,&
          H54,&HØ
MK 4290 REM PLUS.PURP SHAPE S9
KE 4300 DATA &H1C,&H6,&HA,&HA,&H
          A,&HA,&HAØAA,&HAØAA
DM 4310 DATA &HAØAA,&HAØAA,&HA,&
          HA,&HA,&HA,&H5ØØ
PB 4320 REM PLUS.BLU SHAPE S11
FJ 4330 DATA &H1C,&H6,&H5,&H5,&H
          5,&H5,&H5Ø55,&H5Ø55
PK 4340 DATA &H5Ø55,&H5Ø55,&H5,&
          H5,&H5,&H5,&H5ØØ
```

Program 7. Hex War For Amiga

Version by Philip Nelson, Assistant Editor

```
' Hex War for 512K Amiga◂
◂
CLEAR ,25ØØØ◂
CLEAR ,65536&◂
◂
Restart:◂
GOSUB Setup◂
◂
mainloop:◂
GOSUB Reveille◂
GOSUB DrawField◂
GOSUB PlaceTroops◂
GOSUB TakeTurn◂
CLS:talk$="Thinking"◂
LOCATE 12,17:PRINT talk$◂
GOSUB talk◂
GOSUB Battle◂
GOSUB Resolve◂
GOSUB Battle◂
GOSUB Prisoners◂
GOSUB Reinforcements◂
GOSUB Outcome◂
pn=1-pn:ft=Ø:pp=Ø◂
GOTO mainloop◂
◂
DrawField:◂
CLS◂
FOR r=11 TO 1 STEP -2◂
FOR c=12-r TO r+26 STEP 4◂
PUT (c*8,r*8),s1Ø◂
NEXT c,r◂
FOR r=13 TO 21 STEP 2◂
FOR c=r-1Ø TO 49-r STEP 4◂
PUT (c*8,r*8),s1Ø◂
NEXT c,r◂
FOR r=12 TO 1 STEP -2◂
FOR c=14-r TO r+28 STEP r+28-(14
-r)-1◂
LOCATE r,c◂
PRINT CHR$(32)◂
NEXT c,r◂
FOR r=13 TO 21 STEP 2◂
FOR c=r-11 TO 53-r STEP 53-r-(r-
11)-1◂
LOCATE r,c◂
PRINT CHR$(32)◂
NEXT c,r◂
FOR j=2 TO 23 STEP 21◂
LOCATE j,12◂
PRINT SPACE$(19)◂
NEXT◂
FOR j=1 TO 12◂
GOSUB 71Ø◂
NEXT◂
j=1◂
IF gn=1 THEN GOSUB 718:j=2:GOSUB
715◂
IF gn=2 THEN GOSUB 718:j=3:GOSUB
715◂
LOCATE 1,1◂
WHILE INKEY$<>"":WEND◂
RETURN◂
71Ø k=cit(j,Ø)◂
l=cit(j,1)◂
x=(k-1)*2+19◂
y=(12-(k+1))*2+3◂
PUT (x*8+3,y*8+3),s2,PSET◂
RETURN◂
715 k=cit(j,Ø)◂
l=cit(j,1)◂
x=(k-1)*2+19◂
y=(12-(k+1))*2+3◂
PUT (x*8,y*8),s3,PSET◂
RETURN◂
718 k=cit(j,Ø)◂
l=cit(j,1)◂
x=(k-1)*2+19◂
y=(12-(k+1))*2+3◂
PUT (x*8,y*8),s4,PSET◂
RETURN◂
◂
TakeTurn:◂
IF nx(pn)<2 THEN RETURN◂
ht=4:hb=4:GOSUB 1ØØØ◂
81Ø mv=Ø:ct=Ø◂
cb=Ø:pk=Ø:k=Ø◂
FOR j=1 TO nx(pn)-1◂
IF army(j,Ø,pn)>Ø AND army(j,6,p
n)<1 THEN◂
k=1◂
j=nx(pn)-1◂
END IF◂
NEXT j◂
IF k=Ø THEN RETURN◂
CheckIt:◂
IF a$=CHR$(27) THEN a$="":RETURN
◂
ReadMouse:◂
IF MOUSE(Ø)<>2 THEN NoFlag◂
' left button clicked twice◂
WINDOW 4,"Speech",(65,7Ø)-(225,1
1Ø),16,1◂
IF TalkFlag=1 THEN◂
talk$="Now I can talk."◂
PRINT talk$◂
TalkFlag=1-TalkFlag◂
GOSUB talk◂
GOTO ClearMouse◂
```

```
END IF◂
IF TalkFlag=Ø THEN◂
talk$="OK, I'll be quiet."◂
PRINT talk$◂
GOSUB talk◂
TalkFlag=1-TalkFlag◂
END IF◂
ClearMouse:◂
WHILE MOUSE(Ø)<>Ø:WEND◂
PRINT "Press button once"◂
PRINT "to continue..."◂
' wait for one click◂
WHILE MOUSE(Ø)<>1:WEND◂
' purge keyboard, too◂
WHILE INKEY$<>"":WEND◂
WINDOW CLOSE 4◂
NoFlag:◂
qv=Ø:a$=INKEY$:IF a$="" THEN Rea
dMouse◂
IF UCASE$(a$)="Q" THEN◂
GetOut:◂
WINDOW CLOSE 3◂
SCREEN CLOSE 1◂
WINDOW 1,"Hex War",,31,-1◂
WINDOW OUTPUT 1◂
CLEAR ,25ØØØ◂
END◂
END IF◂
IF a$=CHR$(3Ø) THEN qv=3:GOTO Co
deit◂
IF a$=CHR$(31) THEN qv=7:GOTO Co
deit◂
IF a$=CHR$(28) THEN qv=1:GOTO Co
deit◂
IF a$=CHR$(29) THEN qv=5◂
Codeit:◂
j=qv-128*(a$=" ")◂
IF j=Ø THEN CheckIt◂
IF (j AND 128) THEN 11ØØ◂
IF (j AND 1)=Ø THEN CheckIt◂
j=(j-1)/2◂
IF j AND 1 THEN bl=hb+j-2:tl=ht
ELSE tl=ht+1-j:bl=hb◂
IF tl<Ø OR tl>8 THEN CheckIt◂
IF bl<Ø OR bl>8 THEN CheckIt◂
sl=tl+bl:IF sl<4 OR sl>12 THEN C
heckIt◂
hb=bl:ht=tl:GOSUB 1ØØØ◂
LOCATE 1,1◂
FOR z=1 TO 6◂
PRINT SPACE$(8)◂
NEXT◂
qn=map(ht,hb,Ø)◂
IF qn=Ø THEN LOCATE 1,1:GOTO Che
ckIt ELSE ql=map(ht,hb,1)-1◂
LOCATE 1,1◂
PRINT USING "####";qn:PRINT"----
--"◂
FOR j=Ø TO 3◂
PRINT USING "####";army(qn,j,ql)
◂
NEXT◂
LOCATE 1,1◂
GOTO CheckIt◂
◂
1ØØØ sx=146+16*(ht-hb)◂
sy=21Ø-16*(ht+hb)◂
IF map(ht,hb,2)=1 THEN◂
PUT (ox,oy),s8◂
PUT (sx,sy),s7◂
ox=sx:oy=sy◂
pp=1◂
RETURN◂
END IF◂
IF pp THEN◂
PUT (ox,oy),s7◂
PUT (sx,sy),s8◂
ox=sx:oy=sy◂
pp=Ø◂
RETURN◂
END IF◂
IF ft THEN◂
PUT (ox,oy),sB◂
PUT (sx,sy),s8◂
ox=sx:oy=sy◂
RETURN◂
END IF◂
PUT (sx,sy),s8◂
ox=sx:oy=sy:ft=1◂
RETURN◂
◂
11ØØ IF pk=1 THEN 12ØØ◂
IF map(ht,hb,1)<>pn+1 OR map(ht,
hb,Ø)=Ø THEN 81Ø◂
an=map(ht,hb,Ø)◂
IF army(an,6,pn)<>Ø THEN 81Ø◂
pk=1:ct=ht:cb=hb◂
cs=army(an,Ø,pn):SOUND 11ØØ,1Ø◂
talk$=STR$(cs)+CHR$(32)+"roahboh
ts.":GOSUB talk◂
GOTO CheckIt◂
◂
12ØØ j=(ht=ct) AND (hb=cb)◂
IF j AND mv=Ø THEN 81Ø◂
IF j AND mv>Ø THEN 142Ø◂
ax=army(map(ht,hb,Ø),Ø,pn)◂
IF (ax>me AND cs>me) OR (map(ht,
hb,1)-1=1-pn AND ax>Ø) THEN Chec
kIt◂
dt=ABS(ct-ht)◂
db=ABS(cb-hb)◂
tl=db+dt◂
IF NOT (tl=1 OR (ct+cb=ht+hb AND
dt=1)) THEN CheckIt◂
mg=map(ht,hb,Ø)◂
IF mg=Ø THEN 13ØØ◂
FOR j=Ø TO 3◂
army(mg,j,pn)=army(mg,j,pn)+army
(an,j,pn)◂
army(an,j,pn)=Ø◂
NEXT◂
army(mg,6,pn)=1◂
map(ct,cb,Ø)=Ø◂
cs=army(mg,Ø,pn)◂
an=mg:mv=mm+1◂
GOTO 138Ø◂
◂
13ØØ nB=map(ht,hb,1)-1◂
mv=mv+1◂
IF n8<>pn THEN mv=mv+1◂
map(ct,cb,Ø)=Ø◂
map(ht,hb,Ø)=an◂
map(ht,hb,1)=pn+1◂
army(an,4,pn)=ht◂
army(an,5,pn)=hb◂
IF mv>=mm THEN army(an,6,pn)=1◂
k=Ø◂
FOR j=-1 TO 1 STEP 2◂
j1=ht+j:j2=hb+j:j3=hb-j◂
IF j1<Ø OR j1>8 THEN 134Ø◂
IF map(j1,hb,Ø)>Ø THEN◂
IF map(j1,hb,1)=2-pn THEN◂
k=1:j=1:GOTO 136Ø◂
END IF◂
END IF◂
134Ø IF j2<Ø OR j2>8 THEN 135Ø◂
IF map(ht,j2,Ø)>Ø THEN◂
IF map(ht,j2,1)=2-pn THEN◂
k=1:j=1:GOTO 136Ø◂
END IF◂
END IF◂
135Ø IF j3<Ø OR j3>B OR j1<Ø OR
j1>B THEN 136Ø◂
IF map(j1,j3,Ø)>Ø THEN◂
IF map(j1,j3,1)=2-pn THEN k=1:j=
1◂
END IF◂
136Ø NEXT j◂
IF k=1 THEN army(an,6,pn)=1:mv=m
m+1◂
138Ø a=pn:j=ct◂
k=cb:c=Ø:d=Ø◂
GOSUB 183Ø◂
j=ht:k=hb◂
c=cs:d=army(an,6,pn)◂
GOSUB 1B3Ø◂
ct=ht:cb=hb◂
IF mv<mm THEN CheckIt◂
142Ø army(an,6,pn)=1◂
j=ht:k=hb◂
c=cs:d=1◂
GOSUB 183Ø◂
GOTO 81Ø◂
◂
◂
15ØØ RESTORE Strengths◂
FOR j=Ø TO 1◂
nx(j)=5◂
FOR k=1 TO 4◂
READ a,b,c◂
army(k,Ø,j)=a◂
army(k,4,j)=b◂
army(k,5,j)=c◂
map(b,c,Ø)=k◂
map(b,c,1)=j+1◂
NEXT k,j◂
Strengths:◂
DATA 64,2,8,64,3,7,64,4,5,6,64,6,6
◂
DATA 64,2,2,64,3,2,64,5,1,64,6,Ø
◂
FOR j=Ø TO 1◂
FOR k=Ø TO 2Ø◂
a=INT(RND(1)*k*3)◂
FOR l=1 TO 5◂
a=a+INT(RND(1)*21-8)◂
NEXT l◂
IF a<16 THEN a=Ø ELSE a=(a+k*8)
AND 254◂
fq(k,j)=a◂
NEXT k,j◂
RETURN◂
◂
PlaceTroops:◂
FOR j=Ø TO 8◂
FOR k=Ø TO 8◂
a=map(j,k,1)◂
IF a THEN a=a-1:GOSUB 18ØØ◂
NEXT k,j◂
FOR a=Ø TO 1◂
e=13+a*12:f=a*22◂
dx=2-4*a:d=Ø◂
FOR j=Ø TO 8◂
c=fq(j,a):GOSUB 184Ø◂
e=e+dx*2◂
IF j>3 THEN f=f+dx:e=e-dx◂
NEXT j,a◂
IF pn THEN◂
PUT (28Ø,16Ø),s5,PSET◂
ELSE◂
PUT (28Ø,16Ø),s6,PSET◂
END IF◂
RETURN◂
◂
18ØØ b=map(j,k,Ø)◂
c=army(b,Ø,a)◂
d=army(b,6,a)◂
183Ø e=(j-k+1Ø)*2-1◂
f=(13-j-k)*2+1◂
184Ø IF a THEN◂
PUT (e*8+1,f*8),s5,PSET◂
LOCATE f+a+1,e+1◂
PRINT SPACE$(2);◂
GOTO 185Ø◂
END IF◂
PUT (e*8+1,(f+1)*8),s6,PSET◂
LOCATE f+a+1,e+1◂
PRINT SPACE$(2);◂
185Ø IF c=Ø THEN RETURN◂
LOCATE f+a+1,e+1◂
PRINT RIGHT$("Ø"+HEX$(c),2);◂
IF d AND a=Ø THEN◂
LOCATE f+2,e+1◂
PRINT SPACE$(2);◂
PUT (e*8+1,(f+1)*8+1),s11◂
RETURN◂
END IF◂
IF d AND a=1 THEN◂
LOCATE f+1,e+1◂
PRINT SPACE$(2);◂
PUT (e*B+1,f*B+1),s9◂
END IF◂
RETURN◂
◂
Reveille:◂
```

```
sw=Ø:e=nx(pn)-1◄
IF e<1 THEN RETURN◄
FOR j=1 TO e-1◄
IF army(j,Ø,pn)>=1 THEN 197Ø◄
t=army(j,4,pn)◄
b=army(j,5,pn)◄
IF map(t,b,Ø)=j THEN map(t,b,Ø)=
Ø◄
FOR k=j TO e◄
FOR l=Ø TO 6◄
army(k,l,pn)=army(k+1,l,pn)◄
army(k+1,l,pn)=Ø◄
NEXT l◄
t=army(k,4,pn)◄
b=army(k,5,pn)◄
map(t,b,Ø)=k◄
NEXT k◄
nx(pn)=nx(pn)-1◄
j=e: sw=1◄
197Ø NEXT j◄
IF sw THEN Reveille◄
FOR j=1 TO e◄
army(j,Ø,pn)=army(j,Ø,pn)+army(j
,2,pn)◄
army(j,2,pn)=army(j,3,pn)◄
army(j,3,pn)=Ø◄
army(j,6,pn)=Ø◄
NEXT j◄
k=nx(1-pn)◄
FOR j=1 TO k◄
army(j,6,1-pn)=Ø◄
NEXT◄
GOSUB 24ØØ◄
2Ø5Ø IF bp>Ø THEN◄
FOR j=Ø TO 1◄
FOR k=1 TO bp◄
a=BTL(k,j,Ø)◄
army(a,6,j)=army(a,6,j)+1◄
NEXT◄
NEXT◄
END IF◄
RETURN◄
◄
Battle:◄
GOSUB 24ØØ◄
a=nx(Ø)◄
IF nx(1)>a THEN a=nx(1)◄
FOR j=Ø TO 1◄
FOR k=1 TO a◄
army(k,6,j)=Ø◄
NEXT k,j◄
GOSUB 2Ø5Ø◄
RETURN◄
◄
Prisoners:◄
FOR j=Ø TO 1◄
a=1-j◄
b=nx(j)-1◄
FOR k=1 TO b◄
IF army(k,Ø,j)>=1 THEN 228Ø◄
fq(2,a)=fq(2,a)+army(k,2,j)+army
(k,3,j)◄
IF fq(2,a)>255 THEN◄
c=fq(2,a)-255◄
fq(3,a)=fq(3,a)+c◄
fq(2,a)=255◄
END IF◄
fq(6,a)=fq(6,a)+army(k,1,j)◄
IF fq(6,a)>255 THEN◄
c=fq(6,a)-255◄
fq(7,a)=fq(7,a)+c◄
fq(6,a)=255◄
END IF◄
IF map(army(k,4,j),army(k,5,j),Ø
)=k AND map(army(k,4,j),army(k,5
,j),1)=j+1 THEN◄
map(army(k,4,j),army(k,5,j),Ø)=Ø
◄
END IF◄
FOR l=Ø TO 6◄
army(k,l,j)=Ø◄
NEXT◄
228Ø IF army(k,6,j)>=1 THEN 232Ø
◄
fq(4,j)=fq(4,j)+army(k,1,j)◄
army(k,1,j)=Ø◄
IF fq(4,j)>255 THEN◄
c=fq(4,j)-255◄
fq(5,j)=fq(5,j)+c◄
fq(4,j)=255◄
END IF◄
232Ø NEXT k,j◄
RETURN◄
◄

24ØØ bp=Ø◄
FOR j=Ø TO 8◄
j1=(j-4)*(4-j>Ø)◄
j2=B-(j>4)*(4-j)◄
FOR k=j1 TO j2◄
a=map(j,k,Ø)◄
r=map(j,k,1)◄
IF a=Ø OR r=Ø THEN 249Ø◄
IF army(a,Ø,r-1)<1 THEN 249Ø◄
t=j+1◄
b=k: GOSUB 25ØØ◄
b=b-1: GOSUB 25ØØ◄
t=t-1: GOSUB 25ØØ◄
249Ø NEXT k,j◄
RETURN◄
◄
25ØØ IF t<Ø OR b<Ø OR t>8 OR b>8
THEN RETURN◄
pa=map(t,b,Ø)◄
IF pa=Ø THEN RETURN◄
IF map(t,b,1)=r THEN RETURN◄
IF army(pa,Ø,2-r)<1 THEN RETURN◄
bp=bp+1◄
BTL(bp,r-1,Ø)=a◄
BTL(bp,2-r,Ø)=pa◄
RETURN◄
◄
Resolve:◄
IF bp=Ø THEN RETURN◄
FOR j=1 TO bp◄
FOR k=Ø TO 1◄
a=1-k◄
an=BTL(j,k,Ø)◄
ax=army(an,Ø,k)◄
ht=army(an,6,k)◄
ct=INT(ax/ht)+1◄
BTL(j,a,1)=INT(ct*ka+1)◄
BTL(j,a,2)=INT(ct*kb+1)◄
BTL(j,a,3)=INT(ct*kc+1)◄
NEXT k,j◄
FOR j=1 TO bp◄
jØ=BTL(j,Ø,Ø)◄
j1=BTL(j,1,Ø)◄
GOSUB 31ØØ◄
army(jØ,Ø,Ø)=army(jØ,Ø,Ø)-a*BTL(
j,Ø,1)◄
army(j1,Ø,1)=army(j1,Ø,1)-b*BTL(
j,1,1)◄
GOSUB 31ØØ◄
c=a*BTL(j,Ø,2)◄
army(jØ,Ø,Ø)=army(jØ,Ø,Ø)-c◄
army(jØ,1,Ø)=army(jØ,1,Ø)+c◄
c=b*BTL(j,1,2)◄
army(j1,Ø,1)=army(j1,Ø,1)-c◄
army(j1,1,1)=army(j1,1,1)+c◄
GOSUB 31ØØ◄
c=a*BTL(j,Ø,3)◄
army(jØ,Ø,Ø)=army(jØ,Ø,Ø)-c◄
army(jØ,3,Ø)=army(jØ,3,Ø)+c◄
c=b*BTL(j,1,3)◄
army(j1,Ø,1)=army(j1,Ø,1)-c◄
army(j1,3,1)=army(j1,3,1)+c◄
NEXT j◄
RETURN◄
◄
Reinforcements:◄
a=1-pn:b=Ø◄
FOR j=Ø TO 8:FOR k=Ø TO 8◄
IF map(j,k,1)=pn+1 THEN b=b+1◄
NEXT k,j◄
fq(1,pn)=fq(1,pn)+b◄
IF fq(1,pn)>255 THEN◄
b=fq(1,pn)-255◄
fq(2,pn)=fq(2,pn)+b◄
fq(1,pn)=255◄
END IF◄
t=4:b=pn*8◄
IF map(t,b,Ø)<>Ø THEN RETURN◄
IF map(t,b,1)=pn+1 THEN◄
fq(Ø,a)=Ø:fq(1,a)=Ø◄
GOTO 3Ø6Ø◄
END IF◄
j=nx(a)◄
IF j>31 THEN RETURN◄
j1=fq(Ø,a)◄
IF j1<1 THEN 3Ø6Ø◄
nx(a)=nx(a)+1◄
map(t,b,Ø)=j◄
map(t,b,1)=a+1◄
army(j,Ø,a)=j1◄
FOR k=1 TO 3◄
army(j,k,a)=Ø◄
NEXT k◄
army(j,4,a)=t◄
army(j,5,a)=b◄
3Ø6Ø FOR k=Ø TO 19◄
fq(k,a)=fq(k+1,a)◄
NEXT k◄
fq(2Ø,a)=Ø◄
RETURN◄
◄
31ØØ a=Ø:FOR m=1 TO 6◄
IF RND(1)<.5 THEN a=a+1◄
NEXT m:b=6-a◄
RETURN◄
◄
32ØØ talk$="press 1 through 5 to
choose seenaireeo."◄
GOSUB talk◄
WINDOW 4,"Scenario: Press 1-5",(
65,7Ø)-(255,12Ø),16,1◄
PRINT "1> Capture capital/far"◄
PRINT "2> Capture capital/near"◄
PRINT "3> Occupy  8/12 cities"◄
PRINT "4> Control 6/12 cities"◄
PRINT "5> Occupy  4Ø/61 hexes"◄

GrabKey:◄
a$=INKEY$:IF a$="" THEN GrabKey◄
gn=VAL(a$)◄
IF gn<1 OR gn>5 THEN GrabKey◄
WINDOW CLOSE 4◄
talk$="seenaireeo"+STR$(gn)+CHR$
(46):GOSUB talk◄
RETURN◄
◄
Outcome:◄
a=Ø:ON gn GOSUB 343Ø,345Ø,348Ø,3
49Ø,358Ø◄
IF a=Ø THEN RETURN◄
en$=c$:ea=a◄
GOSUB DrawField◄
GOSUB PlaceTroops◄
a=ea◄
WINDOW 4,"Outcome",(25,7Ø)-(3ØØ,
12Ø),16,1◄
PRINT "Player "a" wins"◄
MaybeOut:◄
PRINT c$◄
PRINT "Press Q to quit, RETURN t
o play."◄
a$=""◄
WHILE a$=""◄
a$=INKEY$◄
WEND◄
WINDOW CLOSE 4◄
IF UCASE$(a$)="Q" THEN GetOut◄
WINDOW CLOSE 2:WINDOW CLOSE 1◄
CLEAR ,25ØØØ◄
RUN◄
◄
343Ø IF map(cit(2,Ø),cit(2,1),1)
=1 THEN◄
a=2◄
c$="Red captured the capital"◄
GOSUB Announce◄
RETURN◄
END IF◄
GOTO 346Ø◄
◄
345Ø IF map(cit(3,Ø),cit(3,1),1)
=1 THEN◄
```

```
a=2<
c$="Red captured the capital"<
GOSUB Announce<
RETURN<
END IF<
<
3460 IF map(cit(1,Ø),cit(1,1),1)
=2 THEN<
a=1<
c$="Yellow captured the capital"
<
GOSUB Announce<
RETURN<
END IF<
RETURN<
<
3480 l=8:GOTO 3500<
3490 l=6<
3500 c(1)=Ø:c(2)=Ø<
FOR j=1 TO 12:t=cit(j,Ø):b=cit(j
,1)<
r=map(t,b,1):c(r)=c(r)+1<
IF gn=4 THEN<
an=map(t,b,Ø)<
IF r>Ø THEN IF an=Ø OR army(an,6
,r-1)>Ø THEN c(r)=c(r)-1<
END IF<
NEXT j<
IF c(1)=>l THEN<
a=2<
c$="Red captured"+STR$(c(1))+" c
ities"<
GOSUB Announce<
RETURN<
END IF<
IF c(2)=>l THEN<
a=1<
c$="Yellow captured"+STR$(c(2))+
" cities"<
GOSUB Announce<
END IF<
RETURN<
<
3580 c(1)=Ø:c(2)=Ø<
FOR j=Ø TO 8:FOR k=Ø TO 8<
r=map(j,k,1):c(r)=c(r)+1<
NEXT k,j<
IF c(1)=>40 THEN<
a=2<
c$="Red occupies"+STR$(c(1))+" h
exes"<
talk$="REHD AA4KYUWPAYZ":SAY tal
k$<
talk$=STR$(c(1))+"hehxes"<
GOSUB talk:RETURN<
END IF<
IF c(2)=>40 THEN<
a=1<
c$="Yellow occupies"+STR$(c(2))+
" hexes"<
talk$="YEHLOH AA4KYUWPAYZ":SAY t
alk$<
talk$=STR$(c(2))+"hehxes"<
GOSUB talk:RETURN<
END IF<
RETURN<
<
Setup:<
DEFINT s<
SCREEN 1,320,200,2,1<
' open window 3 with no <
' gadgets or title bar<
WINDOW 1,"",(Ø,Ø)-(311,25),16,1<
WINDOW 3,"",(Ø,Ø)-(311,185),16,1
<
WINDOW OUTPUT 3<
PALETTE Ø,Ø,Ø,Ø<
PALETTE 1,.5,1,1<
PALETTE 2,1,Ø,Ø<
PALETTE 3,1,1,.1<
WIDTH 40<
CLS<
DIM Voice%(8)<
RESTORE VoiceData<
FOR j=Ø TO 8<
READ Voice%(j)<
NEXT<
RESTORE<
' speech will be synchronous<
VoiceData:<
DATA 110,Ø,170,Ø,222ØØ,64,10,1,Ø
<
talk$="Welcome to Hex War."<
LOCATE 13,11<
PRINT talk$<
GOSUB talk<
Temp$="Click button twice to tur
n"<
LOCATE 15,8:PRINT Temp$<
Demp$=" speech off or on during
game."<
LOCATE 16,6:PRINT Demp$<
talk$=Temp$+Demp$<
GOSUB talk<
' hex shape<
LINE (Ø,Ø)-(2,Ø):LINE (13,Ø)-(15
,Ø)<
LINE (Ø,1)-(3,1):LINE (12,1)-(15
,1)<
LINE (3,2)-(12,2):LINE (4,3)-(11
,3)<
FOR j=4 TO 11<
LINE (6,j)-(9,j)<
NEXT<
LINE (4,12)-(11,12):LINE (3,13)-
(12,13)<
LINE (Ø,14)-(3,14):LINE (12,14)-
(15,14)<
LINE (Ø,15)-(2,15):LINE (13,15)-
(15,15)<
DIM s1Ø(225)<
GET (Ø,Ø)-(15,15),s1Ø<
PUT (Ø,Ø),s1Ø<
' cursor shape<
FOR k=Ø TO 1<
GOSUB Bracket<
PAINT (32,42),2+k,1<
LINE (3Ø,35)-(48,44),Ø,bf<
LINE (32,32)-(47,47),Ø,bf<
GOSUB Bracket<
IF k=Ø THEN<
DIM s8(4ØØ)<
GET (26,26)-(53,49),s8<
PUT (26,26),s8<
END IF<
IF k=1 THEN<
DIM s7(4ØØ)<
GET (26,26)-(53,49),s7<
PUT (26,26),s7<
END IF<
NEXT k<
GOTO Blip<
Bracket:<
PUT (16,32),s1Ø<
PUT (32,48),s1Ø<
PUT (48,32),s1Ø<
PUT (32,16),s1Ø<
RETURN<
Blip:<
' city shape<
FOR j=Ø TO 1<
LINE (2,j)-(7,j)<
LINE (2,j+8)-(7,j+8)<
LINE (j,2)-(j,7)<
LINE (j+8,2)-(j+8,7)<
NEXT<
PSET (1,1):PSET (8,1)<
PSET (1,8):PSET (8,8)<
DIM s2(1ØØ)<
GET (Ø,Ø)-(9,9),s2<
PUT (Ø,Ø),s2<
' capital shape<
FOR k=Ø TO 1<
FOR j=Ø TO 3<
LINE (4,j)-(11,j),2+k<
LINE (4,j+12)-(11,j+12),2+k<
LINE (1,4+j)-(14,4+j),2+k<
LINE (1,8+j)-(14,8+j),2+k<
NEXT<
LINE (3,6)-(5,9),Ø,bf<
LINE (6,3)-(9,5),Ø,bf<
LINE (6,1Ø)-(9,12),Ø,bf<
LINE (1Ø,6)-(12,9),Ø,bf<
IF k=1 THEN<
DIM s3(225)<
GET (Ø,Ø)-(15,15),s3<
PUT (Ø,Ø),s3<
END IF<
IF k=Ø THEN<
DIM s4(225)<
GET (Ø,Ø)-(15,15),s4<
PUT (Ø,Ø),s4<
END IF<
NEXT k<
'army shape<
FOR j=Ø TO 14<
LINE (7,Ø)-(j,7),3<
NEXT<
FOR j=4 TO 1Ø<
LINE (7,2)-(j,5),Ø<
NEXT<
DIM s5(64)<
GET (Ø,Ø)-(14,7),s5<
PUT (Ø,Ø),s5<
' other army shape<
FOR j=Ø TO 14<
LINE (7,7)-(j,Ø),2<
NEXT<
FOR j=4 TO 1Ø<
LINE (7,5)-(j,2),Ø<
NEXT<
DIM s6(64)<
GET (Ø,Ø)-(14,7),s6<
PUT (Ø,Ø),s6<
' crosses<
FOR k=Ø TO 1<
FOR j=Ø TO 1<
LINE (Ø,2+j)-(13,2+j),2+k<
LINE (1Ø+j,Ø)-(1Ø+j,5),2+k<
LINE (2+j,Ø)-(2+j,5),2+k<
NEXT<
FOR j=Ø TO 1<
LINE (6+j,Ø)-(6+j,5),Ø<
NEXT<
IF k=Ø THEN<
DIM s11(15Ø)<
GET (Ø,Ø)-(13,13),s11<
PUT (Ø,Ø),s11<
END IF<
IF k=1 THEN<
DIM s9(15Ø)<
GET (Ø,Ø)-(13,13),s9<
PUT (Ø,Ø),s9<
END IF<
NEXT k<
DIM army(31,6,1),BTL(64,1,3)<
DIM map(9,9,2),fq(2Ø,1),nx(1),c(
2)<
cn=12:DIM cit(cn,1)<
RANDOMIZE TIMER<
<
pn=1:me=31<
mm=3 ' Maximum number of moves<
ka=1/48:kb=1/48:kc=1/32<
RESTORE Whatsit<
FOR j=1 TO cn<
FOR k=Ø TO 1<
READ cit(j,k)<
NEXT<
map(cit(j,Ø),cit(j,1),2)=1<
NEXT<
Whatsit:<
DATA 8,4,Ø,4,8,Ø,Ø,8,4,Ø,4,8<
DATA 5,5,3,3,6,3,2,5,5,2,3,6<
CLS<
GOSUB 32ØØ<
CLS<
GOSUB 15ØØ<
RETURN<
<
Announce:<
talk$=c$<
<
talk:<
IF TalkFlag=Ø THEN SAY TRANSLATE
$(talk$),Voice%<
RETURN<
<
```

Leader Board For The 64

David Florance
Programming Assistant

Requirements: Commodore 64 (or Commodore 128 in 64 mode) with a disk drive and a joystick. Versions for the Amiga and Atari ST are planned.

The spring and summer months, with their profusion of golfing events, couldn't be a better time for Access Software to have released its new *Leader Board* professional golf simulator. Continuing in the tradition of such earlier popular releases as *Beach-Head* and *Raid Over Moscow*, Access has fashioned a stunning piece of software in this new golfing game.

All who have tested *Leader Board* agree that it has excellent sound and graphics and is a lot of fun to play. *Leader Board* is easy to use, too. Although you probably won't shoot under-par scores during your first 18 holes, we've yet to see someone play the game and not like it. One person trying *Leader Board* for the first time scored a 52 on *one hole* (for you golf novices, that's not very good), and still said he enjoyed the game. Another player, professing to like neither computers nor golf, is considering buying a computer just to play *Leader Board.*

Bruce and Roger Carver, authors of the game, have done an exceptional job on everything from the golfer's swing to the action of the joystick. The program lets you hook, slice, cut, plug, top, and drub—just as in real life. You can even learn to hit the ball straight—if you concentrate.

Leader Board offers three levels of play: novice, amateur, and professional. Start with the novice level to get some practice. You can even move to the driving range for additional practice on your strokes. The program lets you play anywhere from 18 to 72 holes, and there are four courses from which to choose—each with a distinct personality and level of difficulty. Even the wind is a big factor on the professional level of *Leader Board.*

Teeing off on a typical hole in Leader Board, *an exceptional golf simulator for the Commodore 64.*

Good Whooshes And Plops

From one to four players can take part, and scoring is automatic. The sound effects—from the whoosh of the stroke to the plop of the ball landing in a water hazard—are excellent throughout. Even the sound of the ball dropping into the cup is realistic. The movements of the golfer and the ball in flight (and bouncing on the fairway or green) are superb.

Just as in a real game of golf, you'll need some time to get your strokes down. You control the power of your swing and the direction of the ball by pressing the joystick button and moving the stick forward or backward at the right moments.

After playing hundreds of holes, I've concluded that the most important factor in making good scores is selecting the right clubs. *Leader Board*'s manual offers course cards with detailed yardage indicators as well as a chart with normal club distances. These are invaluable. Access Software is also selling additional tournament disks, with four different courses on each disk, for $19.95 each.

The *Leader Board* disk is not copy-protected, so you can make backups for safekeeping. None of the disks work, however, unless a security key is plugged into the computer's cassette port.

I've been a golfer for about 15 years. Maybe it's a coincidence, but when I went to my local course after playing *Leader Board* for several weeks, I had a great round. Who knows? Maybe *Leader Board* is even improving my game.

Leader Board
Access Software, Inc.
2561 South 1560 West
Woods Cross, UT 84087
$39.95

SunDog: Frozen Legacy For Atari ST

David Florance
Programming Assistant

Requirements: Atari ST-series computer with a color monitor; or an Apple II-series computer with at least 64K RAM and a color monitor. The ST version was reviewed.

Certainly one of the most exciting aspects of the future is space exploration. How will it be out there? What will we find, and how will we learn to adapt and go about our everyday existence? Will people carry the same instincts, societal norms, beliefs, and habits into the dark reaches of space? Whatever happens, it's a sure bet that if life's at all similar to *SunDog: Frozen Legacy*, we'll still have to know what's a good deal and what's not, when to beg, when to borrow, and when to...well, rethink our priorities.

SunDog, from Oasis/FTL games, is a first-rate graphics adventure with enough complexity for the seasoned player but simple enough for a beginner to enjoy. First marketed for the Apple II computers, the new Atari ST version features stunning graphics and easy mouse-driven controls.

You start this adventure with a tremendous inheritance left to you by an ambitious uncle who had designs on building a religious colony. Your task, a large one, is to fill his shoes by completing his dream. It's not easy. There's

more than one obstacle in your path. You don't know where the colony was planned, how to pilot a freighter (the *SunDog*), or how to spend the money he left you. Be careful—chances are you'll be mugged, swindled, and/or lost in a vast mountainous continent before you know it. You can even lose your starship and your inheritance if you don't pay attention to business.

Your first task is to find the colony (named Banville). There's an immense amount of ground to cover because *SunDog: Frozen Legacy* encompasses 50 cities on 18 different inhabited planets in 12 star systems.

Next, you'll have to locate and buy all the goods necessary to complete the colony. Although you start off with enough money to purchase the goods, unless you are very adroit, lucky, or both, you'll make some financial mistakes that may exhaust or at least severely deplete your bank account. Don't despair. Just start over and be more careful next time. Remember, no one promised that life in the far reaches of space would be a rose garden.

Beware Of Beggars

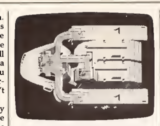
Interaction is an important part of *SunDog: Frozen Legacy*. The shopkeepers and store managers talk to you, and you're expected to respond. Your range of possible responses is limited, though this might be fortunate. If you utter the things that run across your mind while you're trying to finish the mission, you'd probably be hunted down by the galactic security force and executed on sight. So don't be too abrasive.

On the other hand, when a beggar asks you for money, don't give him your last piece of cash. You may need it to transport hastily out of the city. Some of those beggars are ruffians as well, so be prepared to defend yourself. In other words, even in this futuristic setting, the general populace seems still to agree with P.T. Barnum's dictum that a sucker's born every minute. And for the most part, you are considered the latest.

Other characters to watch out for are the many entrepreneurs wandering around or hanging out in the eating establishments. They're more than willing to be your pal. You'll have to be clever to sort out the good and bad deals. These gents can help you as well as harm you.

Popsicle People

To finish the adventure, you have to locate the colonists. They aren't capable of finding you because they're cryogenically frozen. This is when your piloting skills aboard the *SunDog* are put to the test. Setting the course is perhaps the



A top view of the SunDog, *a space freighter in which you search for frozen colonists and a lost colony (Atari ST version).*

most challenging level of the adventure. There are many ways to be successful, and just as many ways to fail. For instance, you can save time by traveling at warp speeds, but you'll spend more fuel, and trying to reach warp can be dangerous. To be safe, be sure to check for engine damage often. When the *SunDog* is not operating at peak efficiency, you lose time and effort. Buying replacement parts is not difficult, and if you shop around you may be able to find some bargains.

The Atari ST gets a chance to show off with *SunDog: Frozen Legacy*. Stunning visual effects abound, and each level has graphic screens that will amaze you. The detail is enormous. If you've been wondering what a graphics adventure game is like on the new generation of personal computers, *SunDog: Frozen Legacy* is a must. It's a whole different world.

A helpful hint from this Star Freighter Captain. Don't let your eyes fool you. You must still eat and sleep to survive and complete the mission. So take time out to catch a few winks and grab a bite to eat. You'll need all your strength and attention to conquer the challenges of *SunDog*.

SunDog: Frozen Legacy
(Atari ST version) Oasis/FTL Games
P.O. Box 112489
San Diego, CA 92111
(Apple II version) Accolade
Entertainment Software
20863 Stevens Creek Blvd.
Cupertino, CA 95014
$39.95 each

The Goonies And *Zorro*

Karen McCullough

Requirements: The Goonies*—an Apple II-series computer with at least 48K RAM and a disk drive; Commodore 64 or 128 (in 64 mode) with a 1541 disk drive; or an Atari 400/800/XL/XE with at least 48K RAM and a disk drive.* Zorro*—same requirements, except Apple II-series computers must have at least 64K RAM. Color monitor optional but recommended. Joystick required. The Apple versions were reviewed.*

Are you tired of shooting aliens and centipedes in arcade-style computer games? Bored with piloting helicopters and drilling holes in brick walls? Does it seem like you've done it all—dodged the best of them, shot the worst—so that it's all a bit of a drag now? Don't give up yet. Datasoft/IntelliCreations has some new challenges for you: games with a smooth blend of arcade action and adventure-like puzzle-solving.

Help the Goonies (that famous band of adventurers from the movie of the same name) negotiate an underground labyrinth to find pirate treasure and save their parents' home from foreclosure. Or perhaps you'd rather be Zorro, using cunning and a sharp sword to rescue a lovely princess from Sergeant Garcia.

Whichever fantasy you choose, you'll need the standard arcade equipment: a keen eye and quick reflexes. But in these games, there's more to negotiating an underground maze than jumping over cannonballs and dodging bats. What do you do about steampipes that block your path? And how can you scale new heights without a ladder? If you consider it unreasonable to have to think about a problem rather than shoot your way out, you'd better consider another game.

Still interested? Good. These two games have a lot to recommend them: excellent graphics and animation; smooth, fast screen changes; and accurate control. At the start of each session with the Apple version, the program asks you to calibrate your joystick by moving the stick to the right-, left-, top-, and bottommost positions. Thereafter, the game adjusts itself to your stick settings, resulting in control as tight and as precise as found in most coin-operated videogames. (Joystick calibration isn't necessary with the Commodore and Atari versions.)

Inventive Puzzles

One of the best features of these games is their interesting and inventive puzzles. There's a lot happening on each screen, and it may take some time to figure out what it all means. In *The*

ATARI 130XE

ATARI 130XE Super Computer Package

130XE Computer	1027 Printer
1050 Disk Drive	Atariwriter +

Call for individual & super package price

ATARI PRINTER INTERFACES

Uprint A	54.95	MPP 1150	54.95

Supra 1000E Modem 44.95

Atari XM-301 Modem 44.95

ATARI 130-XE SUPER PRINTER PACKAGES

NX-10 Printer & U-Print A 319

Panasonic 1091 & U-Print A 299

Super Printer Packages have no extra shipping charges or credit card surcharges when shipped in Continental USA

ATARI 130XE SOFTWARE

BRODERBUND	
Print Shop	26.95
Karateka	20.95
Print Shop Graph I, II, or III	19.95
Print Shop Comp	27.95
INFOCOM	
See Commodore 64 section for items and prices	
ELECTRONIC ARTS	
Archon II	24.95
Archon	19.95
Seven Cit of Gold	24.95
Skyfox	24.95
Pinball Const	19.95
One on One	24.95
Super Boulder Dash	19.95
Chessmaster 2000	27.95
Racing Destruction	24.95
MICROPROSE	
Silent Service	23.95
Gunship	23.95
Acrojet	23.95
F-15 Strike Eagle	23.95
Kennedy Approach	23.95
Conflict Vietnam	27.95
SSI	
See Commodore 64 section for items and prices	
MISCELLANEOUS 130XE	
Mind Pursuit	23.95
Never Ending Story	23.95
Fooblitzky	27.95
Syncalc	32.95
Typesetter	27.95
Ultima III	37.95
Ultima IV	41.95
Hacker	19.95
Beachead II	27.95
Raid Over Moscow	27.95
Fight Night	23.95
Hardball	23.95
Flight Simulator II	34.95
Letter Perfect	39.95
Data Perfect	39.95
Alternate Reality	27.95
Fleet System II	49.95
Page Designer	23.95
Megafont II	19.95
Rubber Stamp	23.95
Sargon III	34.95
Halley Project	23.95
Synfile	32.95

ATARI 520ST*

Atari 520ST-RGB System ...... Call

Atari 520ST-Monochrome Sys ... Call

SF314DS/DD1 Megabyte Disk Dr ... Call

We warranty all 520ST computers purchased from ComputAbility for ninety days

***Please call for stock availability on Atari ST products before ordering by mail.**

MISCELLANEOUS ST

Gato	34.95
Kings Quest II	34.95
ST Talk	17.95
H&D Base	69.95
Typesetter ST	27.95
Habadex Phone	39.95
Word Invaders	24.95
Mean 18 Golf	34.95
Financial Cookbook	34.95
Brataccus	34.95
Michtron Utilities	41.95
Exploding Fist	34.95
Black Cauldron	27.95
Printmaster ST	27.95
PC Intercom	89.95
Final Word	94.95
Hex	27.95
Mudpies	27.95
Sundog	27.95
Flipside	27.95
Softspool	27.95
VIP Professional	119.95
Ultima II	39.95
Perry Mason	34.95
Degas	27.95
Farenheit 451	34.95
Amazon	34.95
DOS Shell	34.95
Kissed	27.95
9 Princess/Amber	34.95
Hacker	29.95
Dragonworld	34.95
Treasure Island	27.95
Wizard of Oz	27.95
Transylvannia	27.95
Borrowed Time	34.95
Mi Term	34.95
Regent Word	34.95
Regent Spell	34.95
Regent Base	Call
Goldrunner	27.95
Time Bandit	27.95
Zoomracks	49.95
Easy Draw	99.95
Mindshadow	34.95
Phantasie	27.95
Hippopotamus	Call
Supra Hard Drive	Call
Supra 1200 ST Modem	Call
OMI 1200 ST Modem	Call
PC Board Designer	Call
Infocom	See IBM

AMIGA

Call for Hardware and Add-on Peripherals prices

AMIGA SOFTWARE

Hacker	29.95
Mindshadow	29.95
Mastertype	27.95
VIP Professional	139.95
Transylvannia	27.95
Crimson Crown	27.95
Racter	29.95
Analyze	69.95
On-line	49.95
Artic Fox	29.95
Brataccus	34.95
Adventure Const	29.95
Deluxe Video	Call
Deluxe Paint	Call
Infocom	See IBM
Maxicom	39.95
Maxidesk	49.95
Maxiplan	119.95
One on One	29.95
7 Cities/Gold	29.95
Skyfox	29.95
Marble Madness	29.95
Return/Atlantis	29.95
Archon	29.95
Aegis Animator	99.95
Exploding Fist	27.95
Deluxe Print	Call
Music Studio	39.95
Borrowed Time	29.95
Financial Cookbook	34.95

APPLE

LAZER 128 APPLE IIC COMPATIBLE

Call for latest prices

APPLE SOFTWARE

BRODERBUND	
Print Shop	33.95
Print Shop Graphics I, II, or III	17.95
Print Shop Comp	27.95
Karateka	21.95
Carmen Sandiego	25.95
Science Tool Kit	39.95
Bank Street Writer	44.95
Fantavision	34.95
ELECTRONIC ARTS	
Adventure Const	34.95
Archon II	27.95
Bard's Tale	29.95
Imagic Football	24.95
Auto Duel	34.95
Skyfox	27.95
Lords of Conquest	27.95
One on One	27.95
Ultima III	39.95
Ultima IV	39.95
Moebius	39.95
Amnesia	Call
EPYX	
Ballblazer	24.95
Winter Games	24.95
Summer Games II	24.95
World's Great/ Football	24.95
Eidolon	24.95
Apshai Trilogy	24.95
Koronis Rift	24.95
INFOCOM	
See IBM section for items and prices	
MICROPROSE	
See Atari 130XE section for items and prices	
SIR-TECH	
Wizardry/Diam	23.95
Wizardry/Legacy	27.95
Wizardry/Proving	33.95
Wizardry/Werdna	29.95
Wizprint	19.95
SSI	
See Commodore 64 section for items & prices	
APPLE MISCELLANEOUS	
Beachead	23.95
Beachead II	27.95
Gamemaker	34.95
Hacker	27.95
Hardball	24.95
Sundog	27.95
Star League Base	23.95
Microleague Base	29.95
Newsroom	39.95
Clip Art Vol I	20.95
Clip Art Vol II	27.95
Gato	27.95
Kings Quest II	34.95
Dambusters	24.95
Alternate Reality	27.95
Kung Fu Master	25.95
Karate Champ	25.95
Strip Poker	23.95
Math Blaster	34.95
Fight Night	24.95
Crossword Magic	34.95
Typing Tutor III	34.95
Law Of The West	24.95
PSI Trading	24.95
Phantasie II	27.95
Rambo	27.95
Amer Challenge	27.95
Crossword Magic	34.95

IBM PC

Corona PC-400 Compatible .... Call

Corona Portable PC Compatible ... Call

PC Multifunction Boards

We carry the complete line of AST, Hercules, Paradise, STB, and Quadram

Call for current prices

IBM PC SOFTWARE

BRODERBUND	
Print Shop	39.95
Print Shop Graph I	27.95
Bank Street Writer	49.95
Ancient Art of War	29.95
MICROPROSE	
F 15 Strike Eagle	23.95
Kennedy Approach	27.95
Acrojet	27.95
Silent Service	27.95
IBM MISCELLANEOUS	
Mean 18	34.95
Gato	27.95
Wizardry	39.95
Strip Poker	27.95
Jet	34.95
Newsroom	39.95
Super Sunday	24.95
Home Pak	34.95
Alter Ego	34.95
Hacker	29.95
Borrowed Time	27.95
Isgur Portfolio	164.95
Typing Tutor	34.95
INFOCOM	
Deadline	34.95
Starcross	34.95
Zork I	27.95
Zork II or III	29.95
Witness	27.95
Suspended	34.95
Planetfall	27.95
Sorcerer	29.95
Seastalker	27.95
Cutthroats	27.95
Hitchiker	27.95
Suspect	29.95
Wishbringer	27.95
Infidel	29.95
Enchanter	27.95
Spellbreaker	34.95
Mind Forever	34.95
Ballyhoo	27.95
SIERRA	
King's Quest	34.95
King's Quest II	34.95
Ultima II	39.95

COMMODORE

COMMODORE 128

C-128 Computer	Call	1670 Modem	159
1571 Disk Drive	Call	1350 Mouse	42.95
1902 Monitor	Call	1750 512K Expander	Call

COMMODORE 128 SOFTWARE

Swiftcalc w/Side	49.95	Superscript 128	59.95
Wordwriter + Spell	49.95	Perfect Writer	49.95
Data Manager II	49.95	Sylvia Porter-128	49.95
Fleet System III	54.95	Matrix	39.95
Mach 128	34.95	Perfect Filer	49.95
Superbase 128	69.95	Paperback 128	34.95
King's Quest II	34.95	Viziwrite 128	Call
Home Pak	34.95	Vizistar 128	Call

COMMODORE 64

C-64 Computer	Call	1670 Modem	159
1541 Disk Drive	Call	1802 Monitor	Call
1660 Modem	49.95		

COMMODORE 64/128

SUPER PRINTER PKGS.

NX-10 & Xetec Supergraphic....329

Panasonic 1091 & Xetec Supergraphic 309

Legend 1080 & Xetec Supergraphic 269

Super Printer Packages have no added shipping or charge card surcharges when shipped in Continental USA

While Supplies Last

B.I. 80 Column Card 59.95

XETEC Super Graphic 69.95

Cardco G-Wiz 54.95

GENERAL HARDWARE

star micronics inc

THE POWER BEHIND THE PRINTED WORD.

NX-10	**Call**
SG-10	**Call**
SG-15	369
SD-10	339
SD-15	449
SR-10	489
SR-15	Call
NL-10	Call
NL-10C	Call
PRINTERS	
Panasonic 1091	239
Legend 808	169
Legend 1080	209
Powertype	309
Juki 5510	389
Epson	Call
Panasonic 1080	215
PRINTER BUFFERS	
Microfazer	From 169
U-Buff 16K	79.95
U-Buff 64K	99.95
MODEMS	
Volksmodem 1200	189
Prometheus 1200	299
Maxwell 1200	Call
Maxwell 2400	Call
Prometheus 2400	Call
MONITORS	
Thommson RGB	289
Commodore 1802	Call
Commodore 1902A	289
Gold Star	Call
Amdek	Call
Panasonic 1300DT	279
NEC 1225	139

COMMODORE 64 SOFTWARE

ACCESS	
Mach V Cart	21.95
Leader Board	27.95
SSI	
Battle of Antietam	32.95
Fighter Command (No Atari)	37.95
Panzer Grenidier	24.95
USAAF	37.95
Breakthrough/ Ardennes	37.95
Kampgruppe	37.95
Phantasie II (No Atari)	27.95
Broadsides	24.95
Carrier Force	37.95
Comp Ambush	37.95
Field of Fire (No Apple)	24.95
Gemstone Warrior	21.95
Imp Galactum	24.95
Computer Baseball	24.95
Comp Quarterback	24.95
Wizards Crown (No Atari)	27.95
Rings of Zilfin (No Atari)	27.95
Battle Group (No Atari)	39.95
MICROPROSE	
See Atari 130XE section for items and prices	
ABACUS	
Call for Items and Prices	
INFOCOM	
Zork I	24.95
Zork II or III	27.95
Deadline	29.95
Starcross	29.95
Witness	29.95
Planetfall	24.95
Hitchiker	24.95
Enchanter	24.95
Cutthroats	24.95
Sorcerer	29.95
Spellbreaker	29.95
Ballyhoo	27.95
ELECTRONIC ARTS	
Adv Construction	29.95
Mail Order Monster	24.95
Racing Destruction	24.95
Ultima III	37.95
Ultima IV	41.95
Bard's Tale	27.95
Lords of Conquest	27.95
Chessmaster 2000	27.95
Touchdown Ftball	23.95
Ultimate Wizard	24.95
Super Boulderdash	19.95
See Atari 130XE section for rest of items & prices	
EPYX	
Eidolon	24.95
Koronis Rift	24.95
Winter Games	24.95
Apshai Trilogy	24.95
Fast Load Cart	24.95
Program/Tool Kit	29.95
Vorpal Utility	22.95
Multiplan	44.95
MISCELLANEOUS COMMODORE 64	
Freeze Frame	34.95
Print Shop	28.95
Cal-Kit	34.95
Superbase 64	47.95
Karateka	20.95
Hacker	20.95
Gamemaker	27.95
Ultima II	37.95
Karate Champ	25.95
Sticks of Death	20.95
Kung Fu Master	25.95
Paper Clip/Spell	54.95
Consultant	41.95
Spy vs Spy Vol II	23.95
Internat'l Hockey	19.95
Prt Shop Compan	27.95
Prt Shop Graphics	19.95
Gato	20.95
Jet	29.95
Fleet System II	44.95
Printmaster	24.95
Newsroom	34.95
Fontmaster II	34.95
Sports Lib Vol I	19.95

We stock hundreds of programs for the Apple, Atari, C-64 and IBM

If you don't see it here, don't hesitate to call

ComputAbility™

P.O. Box 17882 Milwaukee, WI 53217

ORDER LINES OPEN

Mon-Fri 11 a.m. - 7 p.m. CST

Sat 12 p.m. - 5 p.m. CST

NO SURCHARGE FOR MASTERCARD & VISA

To Order Call Toll Free

800-558-0003

For Technical Info., Order Inquiries, or for Wisc. Orders

414-351-2007

ORDERING INFORMATION: Please **specify system**. For fast delivery send cashier's check or money order. Personal and company checks allow 14 business days to clear. School P.O.'s welcome. **C.O.D. charges are $3.00**. In Continental U.S.A. include $3.00 for software orders, 4% shipping for hardware, minimum $4.00. Master Card and Visa orders please include card #, expiration date and signature. WI residents please include 5% sales tax. HI, AK, FPO, APO, Puerto Rico and Canadian orders, please add 5% shipping, **minimum $5.00**. All other **foreign orders add 15% shipping, minimum $10.00**. All orders shipped outside the Continental U.S.A. are shipped first class insured U.S. mail. If foreign shipping charges exceed the minimum amount, you will be charged the additional amount to get your package to you quickly and safely. All goods are new and include factory warranty. Due to our low prices all sales are final. **All defective returns must have a return authorization number**. Please call (414) 351-2007 to obtain an R.A.# or your return will not be accepted. Prices and availability subject to change without notice.

Goonies, the solution to a problem often requires getting the two characters on the screen to cooperate. In *Zorro*, you may need a special object, lots of curiosity, or just a bit of luck to solve a puzzle.

The two games are similar in concept, but not identical in execution. *The Goonies* has a stronger arcade action feel, as befits its descent from that exciting (if not very memorable) movie. *Zorro* plays more like a graphic adventure. There are objects to collect (some may have magical properties), a town to map, secret passages to find, and visual puzzles to solve. *The Goonies* has a hint sheet to help you figure out solutions; in *Zorro*, you're on your own.

Good as they are, though, neither game is perfect. *Zorro* could benefit from a hint sheet of its own—even with a good-quality color monitor it's difficult to tell what some of the objects are. One screen has something that looks like (but surely isn't?) a Coca-Cola bottle. A lantern, perhaps? A club? We aren't sure.

The Goonies needs a way to allow a player to practice on upper-level screens without going through all the lower ones. It takes some 20 minutes of playing time to get to the last levels; you arrive with only one or two lives left and promptly get zapped. That's acceptable at an arcade where the real object is to entice you to spend another quarter, but in a home computer game, it can be frustrating. It would be nice to have an option similar to the one in *Lode Runner* which allows you to play on any level you wish, but prevents you from setting an official high score without progressing through the levels in proper order. Of course, high scores aren't much of a consideration in *The Goonies* or *Zorro*, since neither game saves these scores—another minor flaw.

One final quibble concerns the lack of an option for keyboard control, particularly in the Apple version. Many a white-collar computer runs games during lunch hour, but wouldn't dare be caught with anything so unprofessional as a joystick hanging out of its side. Most Apple games have a keyboard option for this reason.

Overall, however, *The Goonies* and *Zorro* are attractive games—fun, interesting, and entertaining. Due to the level of difficulty, they're not appropriate for children under ten, but older kids and adults will have a good time with them.

Zorro
The Goonies
Datasoft/IntelliCreations, Inc.
19808 Nordhoff Place
Chatsworth, CA 91311
Commodore and Atari versions $29.95
Apple versions $39.95

Moebius: The Orb Of Celestial Harmony For Apple

James V. Trunzo

Requirements: Apple II-series computer with at least 64K RAM and a disk drive. Commodore 64/128 version scheduled for release by July.

Fresh on the heels of the celebrated release of *Ultima IV*, Origin Systems has produced yet another program worth raving about. *Moebius: The Orb of Celestial Harmony* capitalizes on the current popularity of ninjas by casting the player as a youthful martial arts disciple on a trip through an oriental world full of danger and excitement.

Moebius combines all the familiar elements of a computer role-playing game with an arcade-style combat system that's both challenging and functional. That is, the arcade-style combat wasn't included merely to show off the program's superior graphics. Your ninja disciple must become proficient with both a sword and his bare hands to defeat enemies and gain experience points that heighten various attributes.

The theme of *Moebius* is simple. After learning all he could from his ninja masters, a wayward disciple named Kaimen stole the Orb of Celestial Harmony. This act has brought much suffering to the land of Khantun. Kaimen has set himself up as supreme warlord and is conducting a reign of terror. He has imprisoned the Holy Ones, replacing them with his own evil monks. Monsters roam the land and infest the waters of Khantun. A savior is needed—a warrior who has devoted his life to Moebius and who has trained with the Sword Master, the Martial Arts Master, and the Zen Master (for all is not won through force of arms). You are that savior, the ninja warrior.

Protect Your Karma

Moebius employs a number of single-keystroke commands, much like other Origin games such as *Ultima* and its sequels. Your commands are varied: You can communicate with the many characters you encounter during your travels (even the skeletons of the victims of Kaimen—if you know magic), swing your sword to cut vegetation that blocks your way, use an item in your inventory, throw a shuriken, and much more.

A typically ominous screen from Moebius: The Orb of Celestial Harmony.

The game is made even easier to play by the use of windowing. Windows often pop up on the main screen to offer various options. For example, pressing the C key to communicate opens a window and gives you the choice of asking a character for help, to follow you, to stay and wait for you, or to go away.

Much thought is required to play *Moebius* well. The mystery and intrigue of the Orient permeate the game, and virtuous behavior is often rewarded. You must think of others before yourself to be successful and preserve the purity of your Karma (very important).

Furthermore, strategy, planning, and quick thinking are a must. Poor villagers are afraid of men carrying swords and are averse to helping them; yet, hungry tigers cannot be fought off with your bare hands. It's up to you to decide when to arm yourself and when to trust in your karate skills.

Tiger Teeth And Panda Hair

The realm of magic is not ignored, either. In *Moebius*, however, magic works a little differently than in dungeon adventure games. Magic requires a strong mind, so you must fast and chant special mantras to activate such spells as Speak with the Dead, Waterwalk, or Cure Sickness. Likewise, your mind must be clear to divine the nature of artifacts. When this magic is combined with another component (tiger teeth, beetle pincers, condor feathers, etc.), you can teleport, use ventriloquism, cause paralysis, and invoke many other charms. Incidentally, the magical components must be found and then either purchased or earned. To get panda hair, for example, you must first trap the bear.

Moebius embodies its own unique playing style and feel, and it gives the player an unmistakable sense of the Far

East and Zen philosophy. As a programming effort, it equals anything on the market today; graphics are top-notch, ranging from full-screen images to highly detailed temples; onscreen instructions and help are easy to use and aid in play; and the challenge of solving *Moebius* ensures many hours of enjoyment as you travel through the realms of Earth, Water, Air, and Fire to find and reclaim the Orb of Celestial Harmony.

Moebius: The Orb of
Celestial Harmony
Origin Systems, Inc.
Distributed by Electronic Arts
2755 Campus Drive
San Mateo, CA 94403
$59.95 ©

HOTWARE: Software Best Sellers

This Month	Last Month	Title	Publisher	Remarks	Apple	Atari	Commodore	IBM	Macintosh
Entertainment									
1.		*Karate Champ*	Data East	Martial arts game	•		•		
2.		*Kung Fu Master*	Data East	Martial arts game	•		•		
3.	1.	*Ultima IV*	Origin Systems, Inc.	Fantasy game	•	•	•		
4.		*Silent Service*	Microprose	Submarine simulation	•	•	•	•	
5.		*Flight Simulator II*	SubLogic	Aircraft simulation	•	•	•		
Education									
1.	2.	*Typing Tutor III*	Simon & Schuster	Typing instruction program	•		•	•	•
2.	1.	*Math Blaster!*	Davidson	Introductory math program, ages 6-12	•	•	•	•	
3	5.	*Homework Helper: Math Word Problems*	Spinnaker	Math tutorial, high school level	•		•		
4.		*Color Me:The Computer Coloring Kit*	Mindscape	Children's artistic tool	•		•		
5.	3.	*New Improved MasterType*	Scarborough	Typing instruction program	•	•	•	•	
Home Management									
1.	2.	*The Newsroom*	Springboard	Do-it-yourself newspaper	•		•	•	•
2	1.	*Print Shop*	Brøderbund	Do-it-yourself print shop	•	•	•		
3		*Print Master*	Unison World	Do-it-yourself print shop				•	
4.		*Sylvia Porter's Personal Financial Planner*	Timeworks	Personal financial package	•		•	•	
5.		*Bank Street Writer*	Brøderbund	Word processing program	•	•	•	•	

To Our Readers:

COMPUTE! Publications is a part of the ABC Consumer Magazines group of ABC Publishing, Inc. and recently we consolidated many of our operations and moved our Customer Service Department to the New York ABC headquarters. If you have any questions regarding back issues, disk orders, book orders, or how to place an order, call toll free **1-800-346-6767** New York residents should call 212-887-8525.

If you want to order a subscription to COMPUTE!, COMPUTE!'s GAZETTE, COMPUTE!'s GAZETTE DISK, or the COMPUTE! DISK, call **1-800-247-5470** or in Iowa call 1-800-532-1272.

Our Editorial Offices remain in Greensboro, North Carolina. If you wish to submit an article for publication, write us at COMPUTE! Publications, Inc., P.O. Box 5406, Greensboro, NC 27403.

We thank you for your interest and continued support of COMPUTE! Publications.

COMPUTE! Publications,Inc. abc

Part of ABC Consumer Magazines, Inc.
One of the ABC Publishing Companies

COMPUTE!'s Author Guide

Most of the following suggestions serve to improve the speed and accuracy of publication. COMPUTE! is primarily interested in new and timely articles on the Commodore 64/128, Atari, Apple, IBM PC/PCjr, Amiga, and Atari ST. We are much more concerned with the content of an article than with its style, but articles should be clear and well-explained.

The guidelines below will permit your good ideas and programs to be more easily edited and published:

1. The upper left corner of the first page should contain your name, address, telephone number, and the date of submission.

2. The following information should appear in the upper right corner of the first page. If your article is specifically directed to one make of computer, please state the brand name and, if applicable, the BASIC or ROM or DOS version(s) involved. In addition, *please indicate the memory requirements of programs.*

3. The underlined title of the article should start about 2/3 of the way down the first page.

4. Following pages should be typed normally, except that in the upper right corner there should be an abbreviation of the title, your last name, and the page number. For example: Memory Map/Smith/2.

5. All lines within the text of the article must be double- or triple-spaced. A one-inch margin should be left at the right, left, top, and bottom of each page. No words should be divided at the ends of lines. And please do not justify. Leave the lines ragged.

6. Standard typing paper should be used (no erasable, onionskin, or other thin paper) and typing should be on one side of the paper only (upper- and lowercase).

7. Sheets should be attached together with a paper clip. Staples should not be used.

8. If you are submitting more than one article, send each one in a separate mailer with its own tape or disk.

9. Short programs (under 20 lines) can easily be included within the text. Longer programs should be separate listings. *It is essential that we have a copy of the program, recorded twice, on a tape or disk.* If your article was written with a word processor, we also appreciate a copy of the text file on the tape or disk. Please use high-quality 10 or 30 minute tapes with the program recorded on both sides. The tape or disk should be labeled with the author's name, the title of the article, and, if applicable, the BASIC/ROM/DOS version(s). Atari tapes should specify whether they are to be LOADed or ENTERed. We prefer to receive Apple programs on disk rather than tape. Tapes are fairly sturdy, but disks need to be enclosed within plastic or cardboard mailers (available at photography, stationery, or computer supply stores).

10. A good general rule is to spell out the numbers zero through ten in your article and write higher numbers as numerals (1024). The exceptions to this are: Figure 5, Table 3, TAB(4), etc. Within ordinary text, however, the zero through ten should appear as words, not numbers. Also, symbols and abbreviations should not be used within text: use "and" (not &), "reference" (not ref.), "through" (not thru).

11. For greater clarity, use all capitals when referring to keys (RETURN, TAB, ESC, SHIFT), BASIC words (LIST, RND, GOTO), and three languages (BASIC, APL, PILOT). Headlines and subheads should, however, be initial caps only, and emphasized words are not capitalized. If you wish to emphasize, underline the word and it will be italicized during typesetting.

12. Articles can be of any length—from a single-line routine to a multi-issue series. The average article is about four to eight double-spaced, typed pages.

13. If you want to include photographs, they should be either 5×7 black and white glossies or color slides.

14. We do not consider articles which are submitted simultaneously to other publishers. If you wish to send an article to another magazine for consideration, please do not submit it to us.

15. COMPUTE! pays between $70 and $800 for published articles. In general, the rate reflects the length and quality of the article. Payment is made upon acceptance. Following submission (Editorial Department, COMPUTE! Magazine, P.O. Box 5406, Greensboro, NC 27403) it will take from four to eight weeks for us to reply. If your work is accepted, you will be notified by a letter which will include a contract for you to sign and return. *Rejected manuscripts are returned to authors who enclose a self-addressed, stamped envelope.*

16. If your article is accepted and you have since made improvements to the program, please submit an entirely new tape or disk and a new copy of the article reflecting the update. We cannot easily make revisions to programs and articles. It is necessary that you send the revised version as if it were a new submission entirely, but be sure to indicate that your submission is a revised version by writing, "Revision" on the envelope and the article.

17. COMPUTE! does not accept unsolicited product reviews. If you are interested in serving on our panel of reviewers, contact the Review Coordinator for details.

Screen Handler 64

Jeffrey Bailey

This useful utility adds some word processor-like features to the Commodore 64's full-screen editor for use within your own BASIC programs. It also works on the Commodore 128 in 64 mode.

Although it's often taken for granted, the Commodore 64 has one of the best full-screen editors in the business. You can easily move the cursor anywhere on the screen, type characters, and make changes wherever you like. Not all computer owners are so lucky.

As good as it is, however, there are some things that the Commodore 64's excellent screen editor can't do. If you were to draw up a wish list for a new screen editor that would be active when a BASIC program requests keyboard input, it might include these features:

- Switchable insert mode.
- Delete key that draws text *into* the cursor.
- Key to move the cursor to the beginning of input.
- Key to move the cursor to the last character typed.
- Key to erase text from the cursor to the end of the line.

That's quite an impressive list, but there's one more feature that's also desirable: a special key to clear out a single field. In this context, a *field* is simply a screen area of a certain size in which the user can type characters. For instance, let's say your BASIC program needs to request the date in the format mm/dd/yy. Each two-digit entry could be defined as a field that's two characters in length. Ideally, the user wouldn't need to type the / character between each field. As each two-digit entry was completed, the cursor would move automatically from the end of one field to the beginning of the next.

To take this concept even further, how about setting up multiple fields at different spots on the screen? If each field works as we've described, it wouldn't be necessary to press RETURN after typing data in each field. During the entry process, the user could move freely from field to field until each one is filled in. Then your program could read the entire screenful of data at once.

It may sound like a tall order, but "Screen Handler 64" makes all of this possible.

Entering The Program

Although Screen Handler 64 is written completely in machine language, no knowledge of machine language is needed to use it. To type in Screen Handler 64, simply use the "MLX" machine language entry program listed elsewhere in this issue. Follow the MLX directions carefully. Here are the addresses you need for MLX:

Starting address: C000
Ending address: C397

You can then begin entering the Screen Handler data from Program 1. When you finish entering all the data, be sure to use the MLX Save option to store at least one copy on tape or disk.

To load Screen Handler into memory, use LOAD"*filename*",8,1 for disk or LOAD"*filename*",1,1 for tape. Although Screen Handler is less than 1K in length, it uses all of the free memory from locations 49152–53247, so you shouldn't do anything to change those locations while the program is active. In addition, you should avoid changing the contents of memory locations 251–255, which Screen Handler uses as well.

Starting Screen Handler

It's quite simple to incorporate Screen Handler into your BASIC programs. After the machine language is in memory, set up a table of the fields you want, then call Screen Handler with a SYS command. Screen Handler takes care of everything else, including moving the input data into string variables. Getting started is actually a three-stage process. Let's look at each one in turn.

The first step after loading Screen Handler is to include the statement SYS 49152 in your BASIC program. This statement, without any extra parameters,

clears the field table that Screen Handler uses internally.

The next step is to set up each individual field, a job that's also done with a SYS. Here's the general format for the command:

SYS 49155,*x*,*y*,*length*,*string$*

Notice that this SYS statement is followed by four parameters. The first two, *x* and *y*, stand for horizontal and vertical screen locations, and can be either numbers or numeric variables. The *x* value must be in the range 0–39, and *y* must be in the range 0–24.

The next parameter, *length*, defines the length of the field in characters. The length parameter can be any numeric value from 1–254. However, keep in mind that the field must not be so long that the screen scrolls when data is typed. If this happens, Screen Handler can't read the input correctly (because the data has moved up one or more lines).

The final parameter, *string$*, can be any type of string variable, from a simple string such as A$ to an array element like FI$(6). Multidimensional arrays like FI$(8,2) can also be used. Screen Handler automatically takes the information from the screen and puts it into this string variable. Note that if the input data is shorter than the field length (if the user types DOG in a ten-character field, for instance), Screen Handler fills the remainder of the string with spaces.

Once you've defined a string within a Screen Handler field, you should not redefine the string until after Screen Handler has been called (see below). This is because Screen Handler sets up pointers to BASIC's string storage space. If you suddenly redefine a string, it may move in memory, confusing the program. After Screen Handler has been called and input has been entered from the keyboard, it is safe to modify the string.

A Better Way To INPUT

Screen Handler permits you to define as many as 50 different fields. When all of the fields are set up, it's time to call Screen Handler. The statement SYS 49158 tells Screen Handler to begin receiving input from the fields you have defined.

At this point, the up and down cursor keys move the cursor from field to field, *not* from line to line as usual. The RETURN key has the same effect as cursor down, moving you forward (down) to the next field. The left and right cursor keys work normally, but only within a field. That is, these keys move the cursor left and right inside the field as usual. But to move to another field, you must press cursor up/down or RETURN.

Pressing CLR/HOME moves the cursor to the beginning of the current field. SHIFT–CLR/HOME erases the field and homes the cursor. The INST/DEL key works normally, but acts only on the current field. The CTRL-I key combination (hold down CTRL and press I) switches Screen Handler in and out of insert mode. The border color changes to indicate when insert mode is active.

Pressing CTRL-D deletes text by pulling it into the present cursor position. CTRL-E erases every character from the cursor to the end of the field. To move the cursor to the last character typed in a field, press CTRL-N.

That takes care of the entire wish list except for the most important part—storing all of the input data in variables. When all of the data is entered in all of the fields, press SHIFT–RETURN. Screen Handler enters the entire screen at once.

Practical Demonstration

Program 2, "Screen Handler Demo," is a simple BASIC program that illustrates how to use Screen Handler. Here's an explanation of how it works.

Line 10 loads Screen Handler from disk, and lines 20–60 create a screen display that outlines the fields visually. The DATA statement in line 70 contains *x*, *y*, and *length* values for defining the fields. Line 90 clears the field table and prepares Screen Handler for use. Lines 100–140 set up a simple loop to read in the values and set them up in Screen Handler's table.

Note that although an array is used in line 130, the program contains no DIM statement. If BASIC encounters an array that was not previously dimensioned, it automatically dimensions the array for 11 elements (numbered 0–10). Since Screen Handler uses some of BASIC's built-in routines, this feature is available as usual. However, it would be better programming practice to explicitly dimension the array variables—and the DIM is required if you want to use more than 11 array elements. In this case, the DIM statement must precede the SYS 49155 statement that assigns the array element to a field. In Program 2, the DIM statement could be placed anywhere before line 100.

Line 160 tells Screen Handler to start accepting data. Again, notice that the string variables must not be modified between the time that the fields are defined (line 130) and Screen Handler is called (line 160). Once the data has all been entered and you press SHIFT–RETURN to accept the fields, lines 170–210 clear the screen and print out the information. At this point in the program, it becomes safe to modify the string variables if needed.

With a little bit of practice, you can write very professional programs in BASIC with Screen Handler's help. Experiment with programs of your own.

Program 1: Screen Handler 64

Please refer to the "MLX" article in this issue before entering the following listing.

```
C000:4C 09 C0 4C 18 C0 4C 73 96
C008:C0 A9 FB 85 FB A9 00 85 38
C010:FD A9 C4 85 FE 85 FC 60 54
C018:A5 FB C9 FE F0 54 18 69 07
C020:05 85 FB 20 FD AE 20 9E 90
C028:B7 8A A0 00 91 F8 20 FD F6
C030:AE 20 9E 87 8A A0 01 91 CA
C038:F8 20 FD AE 20 9E B7 8A DF
C040:C9 FF D0 03 38 E9 01 48 A4
C048:A0 02 91 FB 20 FD AE 20 03
C050:8B B0 85 9E 84 9F 20 A3 E4
C058:B6 6B 20 75 B4 A0 02 B9 90
C060:61 00 91 9E 88 10 F8 A0 C5
C068:03 A5 62 91 FB C8 A5 63 EB
C070:91 FB 60 A5 FB C9 FB D0 EF
C078:01 60 20 92 C2 A9 80 8D 0B
C080:8A 02 A0 00 8C 0E 03 84 03
C088:B6 B1 FD 99 A7 00 C8 C0 BA
C090:05 D0 F6 A9 01 85 02 A4 09
C098:A7 A6 A8 18 20 F0 FF 20 13
C0A0:35 C2 8C A7 02 8E A8 02 17
C0A8:20 00 C1 A0 00 AD 86 02 42
C0B0:91 BD B1 B4 49 80 91 B4 10
C0B8:20 E4 FF C9 00 F0 F9 A0 78
C0C0:00 AA B1 B4 49 80 91 B4 92
C0C8:20 41 C1 A2 01 E4 B6 D0 E6
C0D0:0D C9 1D F0 09 C9 00 F0 5E
C0D8:05 48 20 77 C2 68 20 D2 35
C0E0:FF A9 00 85 D4 20 35 C2 79
C0E8:CC A7 02 D0 08 EC A8 02 4F
C0F0:D0 03 4C 9F C0 E6 02 A5 6A
C0F8:A9 C5 02 B0 A2 4C F9 C1 08
```

```
C100:98 48 8A 48 A9 00 85 B4 C4
C108:85 B5 68 85 FF A2 00 E4 90
C110:FF F0 0F A5 B4 18 69 28 0D
C118:90 02 E6 B5 85 B4 E8 4C B9
C120:0F C1 68 85 FF A5 B4 18 19
C128:65 FF 90 02 E6 B5 85 B4 5E
C130:85 BD A5 B5 18 69 D8 85 93
C138:BE A5 B5 18 69 04 85 B5 D8
C140:60 8A C9 93 D0 05 68 68 DC
C148:4C DE C2 C9 13 D0 05 68 EC
C150:68 4C 93 C0 C9 91 D0 05 D4
C158:68 68 4C D5 C1 C9 11 D0 39
C160:05 68 68 4C F9 C1 C9 9D 5A
C168:D0 06 68 68 4C 22 C2 60 3A
C170:C9 0D D0 05 68 68 4C F9 FD
C178:C1 C9 94 D0 08 20 77 C2 61
C180:68 68 4C 9F C0 C9 14 D0 FB
C188:05 68 68 4C C8 C2 C9 09 68
C190:D0 18 A5 B6 D0 0A E6 B6 D5
C198:EE 20 D0 68 68 4C 9F C0 B0
C1A0:C6 B6 CE 20 D0 68 68 4C 56
C1A8:9F C0 C9 04 D0 06 20 A7 2C
C1B0:C2 A9 00 60 C9 8D D0 05 31
C1B8:68 68 4C F3 C2 C9 05 D0 6B
C1C0:08 20 41 C3 68 68 4C 9F D1
C1C8:C0 C9 0E D0 E6 20 59 C3 1C
C1D0:68 68 4C 9F C0 A5 FD C9 88
C1D8:00 D0 05 A5 FB 18 69 05 A3
C1E0:38 E9 05 85 FD A0 00 A9 10
C1E8:A7 85 71 84 72 B1 FD 91 FF
C1F0:71 C8 C0 05 D0 F7 4C 93 5A
C1F8:C0 A5 FD C5 FB D0 02 A9 33
C200:FB 18 69 05 85 FD A0 00 6C
C208:A9 A7 85 71 84 72 B1 FD 63
C210:91 71 C8 C0 05 D0 F7 AD E8
C218:0E 03 C9 AA D0 01 60 4C E0
C220:93 C0 A9 01 C5 02 90 03 3F
C228:4C 9F C0 C6 02 A9 9D 20 52
C230:D2 FF 4C 9F C0 38 20 F0 BA
C238:FF 98 C9 28 90 04 38 E9 8E
C240:28 A8 60 20 35 C2 20 00 07
C248:C1 A5 B4 85 9E A5 B5 85 83
C250:9F A5 BD 85 AE A5 BE 85 2E
C258:AF A4 A7 A6 A8 20 00 C1 C5
C260:A5 9E 38 E5 B4 85 FF A5 27
C268:A9 38 E5 FF C9 02 B0 03 48
C270:68 68 60 38 E9 02 60 20 0C
C278:43 C2 A8 B1 9E C8 91 9E 5A
C280:88 B1 AE C8 91 AE 88 88 F9
C288:C0 FF D0 EF C8 A9 20 91 46
C290:9E 60 A0 00 A9 DA 85 B4 0A
C298:84 B5 B1 B4 29 7F 91 B4 6E
C2A0:C8 C8 C0 18 D0 F4 60 20 91
C2A8:43 C2 18 69 02 85 FF A0 E0
C2B0:01 B1 9E 88 91 9E C8 B1 C9
C2B8:AE 88 91 AE C8 C8 C4 FF C7
C2C0:D0 EF 88 A9 20 91 9E 60 3B
C2C8:A5 02 C9 01 D0 03 4C 9F B5
C2D0:C0 C6 02 A9 9D 20 D2 FF 56
C2D8:20 A7 C2 4C 9F C0 A6 A8 6B
C2E0:A4 A7 20 00 C1 A0 00 A9 E0
C2E8:20 91 B4 C8 C4 A9 D0 F9 6E
C2F0:4C 93 C0 A5 FB 85 FD A9 8F
C2F8:AA 8D 0E 03 20 F9 C1 A4 3A
C300:A7 A6 A8 20 00 C1 A0 00 64
C308:B1 B4 20 26 C3 91 AA C8 7E
C310:C4 A9 D0 F4 20 F9 C1 A5 DF
C318:FD F0 03 4C FF C2 A5 B6 0D
C320:F0 03 CE 20 D0 60 29 7F 96
C328:C9 20 B0 04 18 69 40 60 3A
C330:C9 40 B0 01 60 C9 60 B0 6E
C338:04 18 69 20 60 18 69 40 6D
C340:60 20 43 C2 C9 00 F0 10 D4
C348:18 69 02 85 FF A0 00 A9 FA
C350:20 91 9E C8 C4 FF D0 F9 6E
C358:60 A6 A8 A4 A7 20 00 C1 98
C360:A4 A9 88 B1 B4 C0 00 F0 6A
C368:29 C9 20 F0 F5 C8 C4 A9 10
C370:D0 01 88 98 48 A4 A7 A6 06
C378:A8 18 20 F0 FF A9 01 85 9B
C380:02 68 85 FF A0 00 A9 1D 49
C388:20 D2 FF C8 E6 02 C4 FF 2A
C390:D0 F4 60 00 00 00 00 00 C9
```

Program 2: Screen Handler Demo

For instructions on entering this listing, please refer to "COMPUTEI's Guide to Typing In Programs" in this issue of COMPUTEI.

```
PB 10 IF LD=0 THEN LD=1:LOAD "
      SCREEN HANDLER",8,1
SP 20 PRINT "{CLR}":PRINT "***
       SCREEN HANDLER ***":PRI
      NT
BQ 30 PRINT "NAME:{5 SPACES}[
      {20 SPACES}]":PRINT
HC 40 PRINT "ADDRESS:
      {2 SPACES}[{20 SPACES}]"
      :PRINT
PG 50 PRINT "CITY/ST:
      {2 SPACES}[{15 SPACES}]
      {2 SPACES}[{2 SPACES}]":
      PRINT
MS 60 PRINT "PHONE:{4 SPACES}1
      -({3 SPACES})-{3 SPACES}
      -{4 SPACES}"
EM 70 DATA 11,3,20,11,5,20,11,
      7,15,30,7,2,13,9,3,18,9,
      3,22,9,4
RK 80 REM *** CLEAR TABLE ***
CA 90 SYS 49152
KM 100 FOR A=1 TO 7
MR 110 READ X,Y,L
DQ 120 REM *** SET UP TABLE **
       *
JB 130 SYS 49155,X,Y,L,A$(A)
MM 140 NEXT
QS 150 REM *** CALL SCREEN HAN
       DLER ***
XD 160 SYS 49158
CJ 170 PRINT "{CLR}"
EB 180 PRINT A$(1)
QA 190 PRINT A$(2)
SQ 200 PRINT A$(3);", ";A$(4)
SJ 210 PRINT "1-(";A$(5);")-";
       A$(6);"-";A$(7)
```
©

Attention Programmers

COMPUTE! magazine is currently laaking for quality articles an Commadare, Atari, Apple, and IBM computers (including the Cammadare Amiga and Atari ST). If yau have an interesting hame application, educatianal program, programming utility, ar game, submit it ta COMPUTEI, P.O. Bax 5406, Greensbara, NC 27403. Or write far a capy af aur "Writer's Guidelines."

Atari Sound Development System

Michael Ryder

This versatile program lets you design sounds onscreen with a joystick and the keyboard, taking advantage of virtually every feature built into the Atari's sound chip—including some features which are rarely exploited. The custom sounds can then be saved on disk, played back, or added to your own BASIC programs. For all Atari 400/800, XL, and XE computers with at least 40K RAM and a disk drive.

Ever since the Atari was first introduced in 1979, its sound capabilities have always played second fiddle to its graphics. In fact, even many Atari owners are unaware that the sound chip inside their computer has switchable high-pass filters, optional 16-bit frequency resolution, and an adjustable clock rate for modifying the frequency range. Part of the problem is that Atari BASIC's SOUND statement doesn't begin to touch these capabilities; they're accessible only with PEEK and POKE or machine language.

The three programs included here, collectively known as the "Atari Sound Development System," make it easier for you to take advantage of these features—or get acquainted with them in the first place. The main program, "Sound Editor," is a utility that puts the full range of Atari sound capabilities at your command with keyboard and joystick controls. It also lets you design sounds with ADSR envelopes—a feature that, as we'll see in a moment, isn't even built into the Atari sound chip.

Two additional programs, "Sound Player" and "Sound Program Writer," let you play back the sounds you create with the Sound Editor or automatically generate stand-alone programs that can be converted into subroutines of your own BASIC programs.

Developing Sounds

To get started, type in and save Program 1 below. The Sound Editor is the main program that lets you develop, modify, save, and load sounds.

When you type RUN, at first you'll see nothing but a black screen and hear a few beeps. The beeps signal that everything is running normally while the Sound Editor sets itself up. After a short delay, you'll see the Main Menu, which leads to several submenus for various functions:

MAIN MENU
1—Develop sounds
2—Save/Load/Del/Dir sound envelopes
3—EXIT PROGRAM
Your choice (1–3): ?

Option 1 is the gateway into the main part of the program. Option 2 leads to the Input/Output Menu. Option 3 stops the program and exits to BASIC. Let's tackle option 1 first, since that's the meat of the Sound Editor.

When you press 1, the Sound Menu pops up:

SOUND MENU
1—Envelope Editor, Voice 0
2—Envelope Editor, Voice 1
3—Envelope Editor, Voice 2
4—Envelope Editor, Voice 3
5—Play Voices Menu
6—Clear all voices
7—MAIN MENU
Your choice (1–7): ?

Options 6 and 7 are fairly obvious: 6 resets all four voices, discarding any existing values that may have been entered, and 7 returns to the Main Menu shown above. The other options let you design, modify, and play sounds in numerous ways.

Options 1–4 let you use a joystick to design individual sound envelopes for any of the four voices. Each Envelope Editor screen shows a graphic display of the current envelope and also indicates the pitch value assigned to that voice (see photos). Since envelopes aren't actually built into the Atari sound chip, but instead are handled by this program, let's backtrack for a moment and explain how they work.

Attack And Retreat

One of the many characteristics which distinguish different sounds is the shape of their *ADSR envelopes*. ADSR stands for *attack, decay, sustain,* and *release*. These are the four stages of volume changes that occur during a sound's duration.

Attack is the initial rise in volume to the sound's peak volume. Decay is the decrease in volume that follows the peak. Sustain is the period in which the sound continues to be audible. And release is the final drop in volume to silence. Photo 1 is a typical ADSR envelope.

By changing the shape of this envelope, you can vary the effect of the sound. For instance, a percussive sound has an almost instantaneous attack, very short decay and

sustain, and a fairly sharp release (Photo 2).

If you pluck a guitar string and let it resonate, the attack is some-

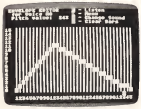


Photo 1: An attack-decay-sustain-release envelope created with the Sound Editor.

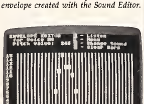


Photo 2: A sharp attack and fast release is typical of percussion instruments.

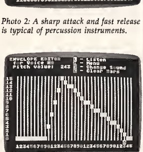


Photo 3: Most musical instruments have a more gentle attack and release.

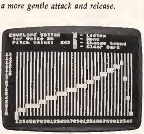


Photo 4: This envelope makes a backward sound.

what less steep and the sustain/release is much more gradual (Photo 3).

Since most real-world sounds have similar envelopes, sounds with a gradual attack and a rapid sustain/release often seem backward and artificial (Photo 4).

You can design almost any kind of ADSR envelope with the Sound Editor. At the bottom of the Envelope Editor screen for each voice is a joystick-controlled cursor. By pushing the joystick left or right, you can move the cursor horizontally to pick a position within the envelope. To set a volume level for that position, press the joystick button, then move the cursor up and down with the stick. When the cursor is at the desired level, press the joystick button again. The level is marked with a white block, and you can move the cursor left or right again to pick the next position.

If you change your mind and want to reset a volume level within the envelope, just move back to that position with the joystick and press the button as before. You can clear out the entire envelope by pressing C (Clear bars).

Other keyboard commands available on the Envelope Editor screens are L (Listen to the envelope), M (return to the Sound Menu), and S (change Sound). If you finish one voice's envelope and want to design an envelope for a different voice, press M for the Sound Menu, then select 1–4 to switch to the other voice's Envelope Editor.

Controlling Sounds

When you press S (change Sound) on an Envelope Editor screen, you get another screen that offers a wide range of control over the POKEY chip, which is responsible for Atari sound. Most of these controls are inaccessible from BASIC without PEEK and POKE. The control screen displays this information:

Modify/Mix Sounds
PRESS TRIGGER TO LISTEN TO ALL VOICES

Switch clock from 64KHz to 15KHz	**:0**
Hi-pass filter on ch.1, clock by 3	**:0**
Hi-pass filter on ch.0, clock by 1	**:0**
Join channels 3 and 2	**:0**
Join channels 1 and 0	**:0**
Clock channel 2 with 1.79 MHz	**:0**
Clock channel 0 with 1.79 MHz	**:0**
Change from 17 to 9 bit poly	**:0**
CHANNEL	**:0**
VOLUME	**:8**
DISTORTION	**:10**
FREQUENCY	**:243**
STATUS (0/1)	**:ON**

—Press X to Exit Back to Editor—

Press the joystick trigger to play the voices; press it again to stop. The X key returns you to the Envelope Editor.

The other features on the control screen let you change various POKEY settings for the voices. At the left and right of this screen, you'll notice a pair of pointers (greater-than and less-than signs). You can move the pointers up and down the screen with the A and Z keys, respectively. This is how you select a certain control you want to change.

The first eight controls (*Switch clock from 64KHz to 15KHz* through *Change from 17 to 9 bit poly*) can be set to either 0 or 1. When the control is set to 0, it is off; when it's set to 1, it's on. For example, to switch on the control for *Hi-pass filter on ch.0, clock by 1*, you'd move the pointers to that line with the A or Z key, then type 1. To turn it off, you'd type 0. You can turn any of the controls on or off in any combination.

The last five controls on this screen (*CHANNEL* through *STATUS*) work a little differently. After selecting one with the pointers, press the space bar. A question mark prompts you to enter a new value. Type in the new value and press RETURN. The allowable ranges are CHANNEL (0–3), VOLUME (0–15), DISTORTION (0–14), FREQUENCY (0–255), and STATUS (0=ON or 1=OFF). These correspond to the parameters in the BASIC SOUND statement, except for STATUS, which turns the current voice on or off so you can mix the different voices.

The first eight controls, however, aren't accessible with the SOUND statement; they POKE values into certain memory locations which directly control the POKEY chip. Perhaps the best way to learn what these controls do is to run the Sound Editor and simply experiment. For a more technical explanation, read the following section.

Playing POKEY

The POKEY chip is accessed through nine memory locations known as AUDF1 (53760), AUDF2 (53762), AUDF3 (53764), AUDF4 (53766), AUDC1 (53761), AUDC2 (53763), AUDC3 (53765), AUDC4 (53767), and AUDCTL (53768). AUDCTL (*AUDio ConTroL*) controls both the four *AUDio Frequency* registers (which set the frequencies of the voices) and the four *AUDio Control* registers (which set the volumes and distortions of the voices). Each of the eight bits in AUDCTL (0–7) controls some aspect of the sound produced. These eight bits correspond to the eight controls in the Sound Editor:

Bit	Decimal	Description
0	1	Switch main clock base from 64 KHz to 15 KHz.
1	2	Insert high-pass filter into channel 2, clocked by channel 4.
2	4	Insert high-pass filter into channel 1, clocked by channel 2.
3	8	Join channels 4 and 3 (16 bit).
4	16	Join channels 2 and 1 (16 bit).
5	32	Clock channel 3 with 1.79 MHz.
6	64	Clock channel 1 with 1.79 MHz.
7	128	Make the 17-bit poly counter into 9-bit poly.

The clock bits (0, 5, and 6) speed up or slow down clock timers, making higher or lower frequency ranges possible. Setting the voices to 1.79 MHz with bits 5 and 6 will produce a much higher sound. The 64 KHz clock will produce lower sounds, and the 15 KHz clock the lowest.

The filter bits (1 and 2), when turned on, allow only frequencies higher than the clock value to pass through.

Bits 3 and 4 each join two of the channels together to permit 16-bit frequency resolution, allowing greater range (nine octaves instead of five) and pitch accuracy. Normally, the POKEY chip uses only 8-bit frequency resolution.

Finally, bit 7 makes the 17-bit poly counter into a 9-bit poly counter. If this bit is set when using distortion, the distortion pattern becomes more obvious.

Playing The Sounds

Let's go back to the Sound Menu, where there's one more option we haven't covered:

SOUND MENU
1—Envelope Editor, Voice 0
2—Envelope Editor, Voice 1
3—Envelope Editor, Voice 2
4—Envelope Editor, Voice 3
5—Play Voices Menu
6—Clear all voices
7—MAIN MENU
Your choice (1-7): ?

After editing all four voices, you might want to hear what they sound like alone or in unison. Option 5 brings up the Play Voices Menu, which offers all 15 combinations of the four voices at a single keystroke:

PLAY VOICES MENU
— Voice Numbers —

A—0,1,2,3	B—0,1,2
C—0,1,3	D—0,1
E—0,2,3	F—0,2
G—0,3	H—1,2,3
I—1,2	J—1,3
K—2,3	L—0
M—1	N—2
O—3	P—PAST MENU

You'll notice, however, that the sound plays very slowly when you use this option. This is because of the memory consumed by the program, BASIC, and the routine that checks which voices are played in each combination. But don't fret—the sounds are much faster (and better) when played with a shorter BASIC program, such as Program 2, Sound Player. This program simply loads and plays any sound created with the Sound Editor.

Program 3, Sound Program Writer, takes a sound created with the Sound Editor, compacts it, and automatically writes a BASIC program to play it. During the compacting phase, all unused bars at the ends of envelopes are chopped off. In addition, a menu asks you to choose which voices should be included, and all unused voices are omitted from the resulting program. So, for instance, if you design a sound that uses only one of the four voices, you can tell Sound Program Writer to discard the unused voices and use only the one you edited. This makes the sound play noticeably faster.

To add the sound to a BASIC program of your own, simply use the program created by Sound Program Writer as a subroutine. It's saved on disk in ASCII format and can be merged into any program with ENTER.

Disk Operations

Before you can play a sound with the Sound Player or convert it into a program with the Sound Program Writer, you have to save it on disk with the Sound Editor. To do this, select option 2 on the Main Menu (Save/Load/Del/Dir sound envelopes). This brings up the Input/Output Menu:

INPUT/OUTPUT MENU
1—Directory of *.SND files
2—Save envelopes file
3—Load envelopes file
4—Delete envelope file
5—MAIN MENU
Choice: ?

Pick option 2 to save a finished sound on disk. When you type in the filename, don't use an extender; the Sound Editor automatically appends the .SND extender to all envelope files. The same is true when loading a file with option 3 or deleting a file with option 4.

Option 1 calls a directory of all files on the disk with the .SND extender, and option 5 returns to the Main Menu. You can also abort any save, load, or delete operation by typing X and pressing RETURN at the filename prompt.

For instructions on entering these listings, please refer to "COMPUTE!'s Guide to Typing In Programs" in this issue of COMPUTE!

Program 1: Sound Editor

```
NC 10 GRAPHICS 0:POKE 82,0:P
      OKE 710,0:POKE 752,1
BA 20 GOTO 3020
LN 30 REM PRIMARY MENU PAGE
IJ 40 COLOR 1:CLOSE #1:OPEN
      #1,4,0,"K:"
OP 50 POKE 89,SCREEN1:POKE 1
      06,SCREEN1+4:POKE DL+5
      ,SCREEN1:GRAPHICS 0:PO
      KE 710,0:POKE 712,0:PO
      KE 752,1
MH 60 POSITION 0,0:? "{Q}
      {38 R}{E}":FOR D=1 TO
      11:POSITION 0,D:? "|":
      POSITION 39,D:? "|":NE
      XT D
CB 70 POSITION 0,12:? "{Z}
      {38 R}{C}"
EL 80 POSITION 9,2:? "
      {3 SPACES}M A I N   M E
      N U{3 SPACES}"
PF 90 POSITION 1,4:? "1 - De
      velop sounds"
HO 100 POSITION 1,5:? "2 - S
       ave/Load/Del/Dir soun
       d envelopes"
AK 110 POSITION 1,7:? "3 - E
       XIT PROGRAM"
```

```
NI 12Ø POSITION 9,9:? "Your choice (1-3) : ?"
NE 13Ø GET #1,K:POSITION 29,9:? CHR$(K)
GH 14Ø IF K<49 OR K>51 THEN POSITION 9,9:? "YOUR CHOICE (1-3):   ":FOR O=1 TO 1Ø:NEXT O:GOTO 12Ø
JN 15Ø ON K-48 GOTO 18Ø,211Ø,17Ø
BD 16Ø GOTO 12Ø
FL 17Ø GRAPHICS Ø:POKE 82,2:END
BL 18Ø POKE 71Ø,Ø:COLOR 1:CLOSE #1:OPEN #1,4,Ø,"K:"
KN 19Ø POKE 89,SCREEN1:POKE 1Ø6,SCREEN1+4:POKE OL+5,SCREEN1:? CHR$(125)
HP 2ØØ POKE 71Ø,Ø:POKE 712,128
DJ 21Ø POSITION Ø,Ø:? "{Q}{38 R}{E}":FOR O=1 TO 13:POSITION Ø,O:? "|":POSITION 39,O:? "|"
AN 22Ø NEXT O:POSITION Ø,14:? "{Z}{38 R}{C}"
KH 23Ø POSITION 13,2:? "SOUND MENU"
LC 24Ø POSITION 5,4:? "1 - Envelope Editor, Voice Ø"
LB 25Ø POSITION 5,5:? "2 - Envelope Editor, Voice 1"
LK 26Ø POSITION 5,6:? "3 - Envelope Editor, Voice 2"
LD 27Ø POSITION 5,7:? "4 - Envelope Editor, Voice 3"
FL 28Ø POSITION 5,8:? "5 - Play Voices Menu"
ND 29Ø POSITION 5,9:? "6 - Clear all voices"
EF 3ØØ POSITION 5,1Ø:? "7 - MAIN MENU"
AE 31Ø POSITION 6,12:? "Your choice (1-7) : ?"
PN 32Ø GET #1,K:POSITION 26,12:? CHR$(K)
AL 33Ø IF K-48<1 OR K-48>7 THEN POSITION 6,12:? "YOUR CHOICE (1-7): ":FOR O=1 TO 1Ø:NEXT O:GOTO 31Ø
EE 34Ø CH=K-49
NE 35Ø ON K-48 GOTO 4ØØ,4ØØ,4ØØ,4ØØ,182Ø,38Ø,4Ø
GG 36Ø GOTO 31Ø
IN 37Ø REM CLEAR ALL VOICES
DC 38Ø FOR X=Ø TO 3:FOR Y=1 TO 35:S(X,Y)=Ø:NEXT Y:NEXT X:GOTO 31Ø
NL 39Ø REM MAIN ROUTINE
JG 4ØØ POKE 89,SCREEN2:POKE 1Ø6,SCREEN2+4:POKE OL+5,SCREEN2:COLOR 16Ø
MP 41Ø POSITION 13,1:? CH
GI 42Ø POSITION 16,2:? SO(CH,Ø)
JN 43Ø FOR A=1 TO 35:R=S(CH,A)
NN 44Ø Y=2Ø-R:X=A+2
LF 45Ø PLOT X,Y
BK 46Ø NEXT A
DD 48Ø OLD=3:POS=OLD
KN 49Ø COLOR 222:PLOT POS,21
LP 5ØØ REM KEYBOARD CHECKER
CD 51Ø POKE 764,255
GI 52Ø S=STICK(Ø):K=Ø
AG 53Ø IF PEEK(764)=255 AND S=15 AND S<>11 AND S<>7 AND STRIG(Ø)<>Ø THEN 52Ø
NN 54Ø IF PEEK(764)=255 THEN 6ØØ
EF 55Ø GET #1,K
GC 56Ø IF CHR$(K)="L" THEN 87Ø
JC 57Ø IF CHR$(K)="M" THEN 175Ø
FF 58Ø IF CHR$(K)="C" THEN 9ØØ
LK 59Ø IF CHR$(K)="S" THEN COLOR 32:PLOT POS,21:GOTO 93Ø
GE 6ØØ IF STRIG(Ø)=Ø THEN 72Ø
NE 61Ø IF S=15 OR S<>11 AND S<>7 AND STRIG(Ø)=1 THEN 52Ø
NB 62Ø IF  NOT ((S=11 AND POS=3) OR (S=7 AND POS=37)) THEN 66Ø
GI 63Ø IF (S=11 AND POS=3) THEN POS=37
DD 64Ø IF (S=7 AND POS=37) THEN POS=3
HC 65Ø GOTO 68Ø
NN 66Ø POS=POS+1*(POS<38 AND S=7)
LP 67Ø POS=POS-1*(POS>3 AND S=11)
HM 68Ø COLOR 222:POKE 53279,Ø:PLOT POS,21
GK 69Ø COLOR 32:PLOT OLD,21
KF 7ØØ OLD=POS
GI 71Ø GOTO 52Ø
EE 72Ø REM FILL IN THE BARS
EA 73Ø FOR S=15 TO Ø STEP -1:SOUND Ø,5Ø,1Ø,S:NEXT S
NC 74Ø P=POS-2:V=S(CH,P):X=POS:Y=2Ø-V:Y2=Y
EA 75Ø S=STICK(Ø):IF S<>14 AND S<>13 AND STRIG(Ø)=1 THEN 75Ø
GD 76Ø IF STRIG(Ø)=Ø THEN 84Ø
GD 77Ø V=V+1*(S=14 AND V<15)
DN 78Ø V=V-1*(S=13 AND V>Ø)
GL 79Ø Y=2Ø-V
DG 8ØØ COLOR 124:PLOT X,Y2
GA 81Ø COLOR 16Ø:PLOT X,Y:Y2=Y
CP 82Ø POKE 53279,Ø
MA 83Ø GOTO 75Ø
NK 84Ø S(CH,P)=V
DN 85Ø FOR S=Ø TO 15:SOUND Ø,255,1Ø,S:NEXT S:SOUND Ø,Ø,Ø,Ø
GD 86Ø GOTO 52Ø
FC 87Ø REM LISTEN
JP 88Ø POKE 53768,BYTE:POKE 5376Ø,SO(CH,Ø):FOR I=1 TO 35:R=S(CH,I):POKE 53761,SO(CH,1)+R:NEXT I
HE 89Ø POKE 53761,16Ø:POKE 764,255:GOTO 51Ø
DN 9ØØ REM CLEAR THE BARS
CG 91Ø COLOR 124:FOR A=35 TO 1 STEP -1:R=S(CH,A):X=A+2:Y=(2Ø-R):PLOT X,Y:NEXT A:COLOR 32:PLOT POS,21
FI 92Ø FOR I=1 TO 35:S(CH,I)=Ø:NEXT I:GOTO 4ØØ
DF 93Ø REM CHANGE DISTORTION
BG 94Ø POKE 82,2:POKE 764,255:POKE 89,SCREEN1:POKE 1Ø6,SCREEN1+4:POKE OL+5,SCREEN1:POKE 752,1:? CHR$(125)
FC 95Ø POKE 53768,Ø:CHAN=CH
AC 96Ø POSITION Ø,Ø:? "Modify/Mix Sounds":POKE 752,1
NN 97Ø POSITION Ø,1:? "PRESS TRIGGER TO LISTEN TO ALL VOICES"
DG 98Ø POSITION 1,2:? "Switch clock from 64KHz to 15KHz{3 SPACES}:";BIT(Ø)
KC 99Ø POSITION 1,3:? "Hi-pass filter on ch.1, clock by 3 :";BIT(1)
MA 1ØØØ POSITION 1,4:? "Hi-pass filter on ch.Ø, clock by 1 :";BIT(2)
NC 1Ø1Ø POSITION 1,5:? "Join channels 3 and 2{14 SPACES}:";BIT(3)
NB 1Ø2Ø POSITION 1,6:? "Join channels 1 and Ø{14 SPACES}:";BIT(4)
PF 1Ø3Ø POSITION 1,7:? "Clock channel 2 with 1.79 MHz{6 SPACES}:";BIT(5)
PG 1Ø4Ø POSITION 1,8:? "Clock channel Ø with 1.79 MHz{6 SPACES}:";BIT(6)
DL 1Ø5Ø POSITION 1,9:? "Change from 17 to 9 bit poly{7 SPACES}:";BIT(7)
DP 1Ø6Ø POSITION 1,1Ø:? "CHANNEL{4 SPACES}: ";CHAN
BB 1Ø7Ø POSITION 1,11:? "VOLUME{5 SPACES}: ";VO(CHAN)
DE 1Ø8Ø POSITION 1,12:? "DISTORTION : ";SO(CHAN,1)/16
EC 1Ø9Ø POSITION 1,13:? "FREQUENCY  : ";SO(CHAN,Ø)
FE 11ØØ POSITION 1,14:? "STATUS(Ø/1): ";:IF STAT(CHAN)=1 THEN ? "ON ":GOTO 112Ø
CB 111Ø ? "OFF"
JC 112Ø POSITION 2,16:? " - PRESS X TO EXIT BACK TO EDITOR - "
JA 113Ø CURX=38:CURY=2:POKE 764,255:OLY=CURY:CLOSE #1:OPEN #1,4,Ø,"K:"
LF 114Ø REM RUN THE MENU
FI 115Ø POSITION Ø,CURY:? ">":POSITION 38,CURY:? "<"
NG 116Ø IF PEEK(764)<>255 THEN 12ØØ
NL 117Ø IF STRIG(Ø)=Ø THEN 157Ø
NL 118Ø GOTO 116Ø
PA 119Ø REM TOP PART (BITS Ø-7)
FJ 12ØØ IF PEEK(764)<>255 THEN GET #1,K
GF 121Ø IF K=88 THEN ? "{CLEAR}":GOTO 4ØØ:REM TO VOICE DEVELOPING SCREEN
NK 122Ø IF CURY<1Ø AND (K=48 OR K=49) THEN 126Ø
ND 123Ø IF K=65 OR K=9Ø THEN 127Ø
ME 124Ø IF K=32 AND CURY>9 THEN 131Ø
```

```
JK 1250 POKE 764,255:GOTO 11
        50
NM 1260 SIT(CURY-2)=K-48:POS
        ITION CURX-1,CURY:?
        SIT(CURY-2):POKE 764
        ,255:GOTO 1150
KN 1270 CURY=CURY-1*(K=65):C
        URY=CURY+1*(K=90):IF
         CURY<2 THEN CURY=14
CN 1280 IF CURY>14 THEN CURY
        =2
LA 1290 POSITION CURX,OLY:?
        CHR$(32):POSITION 0,
        OLY:? CHR$(32):OLY=C
        URY
NE 1300 GOTO 1150
OO 1310 POKE 752,0
OL 1320 TRAP 1320:POSITION 1
        ,18:? "{DEL LINE}";:
        INPUT NUM:TRAP 40000
AF 1330 IF NUM<>INT(ABS(NUM)
        ) OR NUM<0 THEN 1320
OO 1340 ON CURY-9 GOTO 1360,
        1390,1420,1460,1490
AO 1350 REM PICK A CHANNEL
BF 1360 IF NUM>3 THEN POSITI
        ON 1,18:? "
        {DEL LINE}Channel (0
        -3)":FOR O=1 TO 200:
        NEXT O:GOTO 1320
EN 1370 CHAN=NUM:GOTO 1510
PA 1380 REM PICK A VOLUME
ON 1390 IF NUM>15 THEN POSIT
        ION 1,18:? "
        {DEL LINE}Volume (0-
        15)":FOR O=1 TO 200:
        NEXT O:GOTO 1320
DH 1400 VO(CHAN)=NUM:GOTO 15
        10
CH 1410 REM PICK A DISTORTIO
        N
DM 1420 IF NUM>14 THEN POSIT
        ION 1,18:? "
        {DEL LINE}Distortion
         (0-14)":FOR O=1 TO
        200:NEXT O:GOTO 1320
AK 1430 IF NUM/2<>INT(NUM/2)
         THEN NUM=NUM-1:IF N
        UM<0 THEN NUM=0
CA 1440 SO(CHAN,1)=NUM*16:GO
        TO 1510
NI 1450 REM PICK A FREQUENCY
NH 1460 IF NUM>255 THEN POSI
        TION 1,18:? "
        {DEL LINE}Frequency
        (0-255)":FOR O=1 TO
        200:NEXT O:GOTO 1320
JH 1470 SO(CHAN,0)=NUM:GOTO
        1510
PN 1480 REM PICK A STATUS
CB 1490 IF NUM>1 THEN POSITI
        ON 1,18:? "
        {DEL LINE}Status (0=
        OFF 1=ON)":FOR O=1 T
        O 300:NEXT O:GOTO 13
        20
NE 1500 STAT(CHAN)=NUM:GOTO
        1510
CP 1510 POSITION 1,18:? "
        {5 DEL LINE}":POKE 7
        52,1
HN 1520 POSITION 14,10:? CHA
        N:POSITION 14,11:? V
        O(CHAN);"  "
HJ 1530 POSITION 14,12:? SO(
        CHAN,1)/16;"  ":POSI
        TION 14,13:? SO(CHAN
        ,0);"  "
KH 1540 POSITION 14,14:IF ST
        AT(CHAN)=1 THEN ? "O
        N ":GOTO 1560
CJ 1550 ? "OFF"
NN 1560 GOTO 1150
DC 1570 REM TURN ON ALL VOIC
        E CHANNELS
SP 1580 BYTE=0
DD 1590 IF BIT(0) THEN BYTE=
        SYTE+1
DI 1600 IF SIT(1) THEN BYTE=
        BYTE+2
DM 1610 IF SIT(2) THEN SYTE=
        BYTE+4
PC 1620 IF BIT(3) THEN SYTE=
        SYTE+8
CD 1630 IF SIT(4) THEN BYTE=
        BYTE+16
CD 1640 IF BIT(5) THEN SYTE=
        BYTE+32
CK 1650 IF BIT(6) THEN SYTE=
        BYTE+64
FN 1660 IF SIT(7) THEN BYTE=
        SYTE+128
DN 1670 POKE 53768,0:POKE 53
        768,BYTE
SO 1680 FOR X=0 TO 3:IF STAT
        (X)=0 THEN 1700
NN 1690 POKE 53760+(X*2),SO(
        X,0):POKE 53761+(X*2
        ),SO(X,1)+VO(X)
FP 1700 NEXT X
DC 1710 POSITION 0,1:? " PRE
        SS TRIGGER TO TURN O
        FF ALL VOICES "
NK 1720 IF STRIG(0)=1 THEN 1
        720
SC 1730 POSITION 0,1:? " PRE
        SS TRIGGER TO LISTEN
         TO ALL VOICES "
KL 1740 FOR X=0 TO 8:POKE 53
        760+X,0:NEXT X:GOTO
        1150
EA 1750 REM CLEAR SARS
IN 1760 COLOR 124:FOR A=35 T
        O 1 STEP -1:R=S(CH,A
        ):X=A+2:Y=(20-R):PLO
        T X,Y
EP 1770 NEXT A
JL 1780 COLOR 32:PLOT OLO,21
ME 1790 POSITION 13,1:? " "
MN 1800 POSITION 16,3:? "
        {3 SPACES}"
JN 1810 GOTO 180
LD 1820 REM LISTEN MENU
ND 1830 POKE 710,0:COLOR 1:P
        OKE 89,SCREEN1:POKE
        106,SCREEN1+4:POKE O
        L+5,SCREEN1:? CHR$(1
        25):POKE 752,1:? "
        {CLEAR}"
HE 1840 POSITION 0,0:? "{Q}
        {38 R}{E}":FOR O=1 T
        O 14:POSITION 0,O:?
        "|":POSITION 39,O:?
        "|"
EI 1850 NEXT O:POSITION 0,15
        :? "{Z}{38 R}{C}"
AL 1860 POSITION 8,2:? "
        {4 SPACES}PLAY VOICE
        S MENU{4 SPACES}"
OP 1870 POSITION 7,4:? " --
        Voice nuebers --"
KN 1880 POSITION 6,6:? "A -
        0,1,2,3{9 SPACES}B -
         0,1,2"
PN 1890 POSITION 6,7:? "C -
        0,1,3{11 SPACES}D - 0
        ,1"
PH 1900 POSITION 6,8:? "E -
        0,2,3{11 SPACES}F - 0
        ,2"
PD 1910 POSITION 6,9:? "G -
        0,3{13 SPACES}H - 1,2
        ,3"
MN 1920 POSITION 6,10:? "I -
         1,2{13 SPACES}J - 1,
        3"
NF 1930 POSITION 6,11:? "K -
         2,3{13 SPACES}L - 0"
NN 1940 POSITION 6,12:? "M -
         1{15 SPACES}N - 2"
IA 1950 POSITION 6,13:? "O -
         3{15 SPACES}P - PAST
         MENU"
SC 1960 POSITION 8,15:? "
        {W}{20 R}{W}"
CJ 1970 POSITION 8,16:? "|Yo
        ur choice (A-P) : |"
FF 1980 POSITION 8,17:? "
        {Z}{20 R}{C}"
DH 1990 GET #1,K:IF K<65 OR
        K>80 THEN POSITION 8
        ,16:? "|YOUR CHOICE
        (A-P) : |":GOTO 1970
KM 2000 K=K-64:IF K=16 THEN
        180
NN 2010 POKE 53768,BYTE:POKE
         53775,3:POKE 53760,
        SO(0,0):POKE 53762,S
        O(1,0):POKE 53764,SO
        (2,0):POKE 53766,SO(
        3,0)
NN 2020 POKE 559,0:FOR A=1 T
        O 35
IH 2030 POKE 53761,(SO(0,1)+
        S(0,A))*(G(K,1)=1)
IN 2040 POKE 53763,(SO(1,1)+
        S(1,A))*(G(K,2)=1)
IN 2050 POKE 53765,(SO(2,1)+
        S(2,A))*(G(K,3)=1)
JO 2060 POKE 53767,(SO(3,1)+
        S(3,A))*(G(K,4)=1)
OI 2070 NEXT A:POKE 559,34
JK 2080 POKE 53768,0:POKE 53
        761,0:POKE 53763,0:P
        OKE 53765,0:POKE 537
        67,0
JN 2090 POSITION 14,16:? "CH
        OICE : ?":GOTO 1970
DN 2100 REM LOAD/SAVE/DIR/DE
        L
MD 2110 POKE 89,SCREEN1:POKE
         106,SCREEN1+4:POKE
        OL+5,SCREEN1
AA 2120 CLOSE #1:OPEN #1,4,0
        ,"K:"
HH 2130 GRAPHICS 0:POKE 710,
        0:POKE 82,0:POKE 752
        ,1:TRAP 40000:POKE 7
        12,192
IN 2140 ? "{CLEAR} INPUT/OUT
        PUT MENU "
ID 2150 ? "1 - Directory of
        *.SND files"
KL 2160 ? "2 - Save envelope
        s file"
JD 2170 ? "3 - Load envelope
        s file"
AA 2180 ? "4 - Delete envelo
        pe file"
AL 2190 ? "5 - MAIN MENU"
NM 2200 POSITION 0,15:? "CHO
        ICE : ?"
EE 2210 GET #1,K:POSITION 9,
        15:? CHR$(K):IF K<49
         OR K>53 THEN 2130
BH 2220 IF K=53 THEN GOTO 40
DM 2230 ON K-48 GOSUB 2260,2
        350,2830,2530,180
DD 2240 REM DIR,SAVE,LOAD,DE
        L
HI 2250 GOTO 2130
CN 2260 ? "{CLEAR}DIRECTORY
        OF DRIVE 1":POKE 82,
        2:? :POKE 712,4
CJ 2270 CLOSE #2:OPEN #2,6,0
        ,"D1:*.SND":POKE 82,
        2:FLN=0
FD 2280 TRAP 2310:INPUT #2,F
        L$:FLN=FLN+1:IF FL$(
```

```
          5,8)="FREE" THEN 231
          0
BA 2290 FLL$=FL$(3,10):FLL$(
          9,9)=".":FLL$(10,12)
          =FL$(11,13)
EF 2300 ? FLL$:GOTO 2280
JJ 2310 POKE 82,0:? :? FL$:C
          LOSE #2:POKE 82,2
IC 2320 ? :? "Press RETURN t
          o continue...   "
JO 2330 GET #1,K:IF K<>155 T
          HEN 2330
KJ 2340 RETURN
EO 2350 ? "{CLEAR} SAVE A *.
          SND FILE ":POKE 712,
          80
FP 2360 ? :? :? :POKE 752,0
AD 2370 ? "Enter name for fi
          le."
FE 2380 ? " or X to exit."
PO 2390 ? "{3 SPACES}{Q}{22
          R}{E}"
NL 2400 ? "{3 SPACES}|On:fil
          ename.  Extender|"
PJ 2410 ? "{3 SPACES}|automa
          tically attached|"
PE 2420 ? "{3 SPACES}{Z}{22
          R}{C}"
HL 2430 GOSUB 2630:IF FL$="X
          " THEN RETURN
EA 2440 IF PEEK(195)=170 THE
          N 2480
BH 2450 ? :? FL$;" already e
          xists.":? "Do you wa
          nt to rewrite it (Y/
          N) ?"
BH 2460 GET #1,K:IF K<>ASC("
          Y") AND K<>ASC("N")
          THEN 2460
JH 2470 IF K=ASC("N") THEN 2
          350

DH 2480 ? "Okay, saving ";FL
          $;"."
OF 2490 CLOSE #2:OPEN #2,8,0
          ,FL$:PUT #2,BYTE
CA 2500 FOR X=0 TO 3:FOR Y=0
           TO 1:PUT #2,SO(X,Y)
          :NEXT Y:NEXT X
BF 2510 FOR X=0 TO 3:FOR Y=1
           TO 35:PUT #2,S(X,Y)
          :NEXT Y:NEXT X
KJ 2520 RETURN
ME 2530 ? "{CLEAR} DELETE A
          *.SND FILE ":POKE 71
          2,64
FP 2540 ? :? :? :POKE 752,0
HB 2550 ? "Enter file to del
          ete."
FE 2560 ? " or X to exit."
PO 2570 ? "{3 SPACES}{Q}{22
          R}{E}"
DE 2580 ? "{3 SPACES}|On:fil
          ename.  Extender|"
AC 2590 ? "{3 SPACES}|automa
          tically attached|"
PE 2600 ? "{3 SPACES}{Z}{22
          R}{C}"
HL 2610 GOSUB 2630:IF FL$="X
          " THEN RETURN
MO 2620 GOTO 2720
EE 2630 POKE 82,13
OG 2640 POSITION 12,14:? "
          {DEL LINE}";:TRAP 26
          40:INPUT FL$:TRAP 40
          000:IF FL$="" THEN 2
          640
PK 2650 IF FL$="X" THEN RETU
          RN
FO 2660 IF FL$(1,1)<>"D" AND
           FL$(3,3)<>":" THEN
          2640
DD 2670 POKE 82,0:? :TRAP 26
          90:FOR O=4 TO 15:IF
          FL$(O,O)="." THEN PO
          P :GOTO 2640
FD 2680 NEXT O
KO 2690 FL$(LEN(FL$)+1)=".SN
          D"
GC 2700 CLOSE #2:TRAP 2710:O
          PEN #2,4,0,FL$:POKE
          82,0:CLOSE #2
KK 2710 RETURN
HD 2720 IF PEEK(195)<>170 TH
          EN 2750
FO 2730 ? "Sorry, ";FL$;" do
          es not":? "
          {5 SPACES}seem to ex
          ist.  Please try aga
          in...":TRAP 40000
BE 2740 POKE 752,1:POSITION
          12,20:? " Press any
          key ":GET #1,K:GOTO
          2530
GN 2750 ? "Attention: ";FL$;
          " will":? "be utterl
          y destroyed after de
          letion."
LO 2760 REM
KA 2770 ? :? "Press C to con
          tinue or RETURN to a
          bort "
DM 2780 GET #1,K:IF K<>ASC("
          C") AND K<>155 THEN
          2780
HM 2790 IF K=155 THEN 2530
BD 2800 ? CHR$(125):? :? "NO
          W DELETING ";FL$;"..
          ."
DM 2810 XIO 33,#2,0,0,FL$
KM 2820 RETURN
HC 2830 ? "{CLEAR} LOAD A *.
          SND FILE ":POKE 712,
          242
```

Program 2: Sound Player

```
EA 10 GRAPHICS 0:POKE 710,12
        8:POKE 712,8:POKE 82,0
LF 20 DIM SO(3,1),S(3,35),FL
        $(20):OPEN #1,4,0,"K:"
        :POKE 752,1
BC 30 POSITION 0,0:? "
        {8 SPACES}SOUND DEVELO
        PMENT SYSTEM
        {8 SPACES}"
BJ 40 POSITION 0,1:? "
        {14 SPACES}SOUND PLAYER
        {14 SPACES}"
AI 50 ? :? "This program loa
        ds and plays sound
        {6 SPACES}envelopes sa
        ved with the Sound Edi
        tor."
EG 60 ? :? "It can also be u
        sed as a routine in yo
        urown programs."
CF 80 POSITION 3,22:? "-----
        ---- Press any key ---
        ------"
OB 90 POKE 764,255:GET #1,K
AB 100 ? CHR$(125):POSITION
        0,0:? "{8 SPACES}SOUN
        D DEVELOPMENT SYSTEM
        {8 SPACES}":POKE 752,
        0
FL 110 ? :? :? "Enter name f
        or load file."
OD 120 ? "{Q}{25 R}{E}"
IF 130 ? "|On:filename.  Ext
        ender is|"
PI 140 ? "|automatically app
        ended.   |"
PI 150 ? "{Z}{25 R}{C}"
EB 160 POSITION 12,14:INPUT
        FL$:IF FL$="" THEN 16
        0
HO 170 IF FL$(1,1)<>"D" THEN
        100
EB 180 POKE 82,0:? :TRAP 200
        :FOR O=1 TO 15:IF FL$
        (O,O)="." THEN POP :G
        OTO 160
BN 190 NEXT O
GP 200 FL$(LEN(FL$)+1)=".SNO
        "
MM 210 CLOSE #2:TRAP 220:OPE
        N #2,4,0,FL$:CLOSE #2
AP 220 IF PEEK(195)<>170 THE
        N 240
IK 230 ? :? FL$;" does not s
        eem to exist...":POKE
        752,1:POSITION 0,20:
        ? "{6 SPACES}PRESS AN
        Y KEY":GOTO 90
FF 240 ? "Okay, loading ";FL
        $;"."
JA 250 CLOSE #2:OPEN #2,4,0,
        FL$:GET #2,BYTE
AD 260 FOR X=0 TO 3:FOR Y=0
        TO 1:GET #2,Z:SO(X,Y)
        =Z:NEXT Y:NEXT X
PI 270 FOR X=0 TO 3:FOR Y=1
        TO 35:GET #2,Z:S(X,Y)
        =Z:NEXT Y:NEXT X
LP 280 REM ENVELOPE PLAYED H
        ERE
MJ 290 POKE 559,0
EN 300 POKE 53768,BYTE:POKE
        53775,3:POKE 53760,SO
        (0,0):POKE 53762,SO(1
        ,0):POKE 53764,SO(2,0
        ):POKE 53766,SO(3,0)
JN 310 POKE 559,0:FOR A=1 TO
         35
CI 320 POKE 53761,(SO(0,1)+S
        (0,A))
CN 330 POKE 53763,(SO(1,1)+S
        (1,A))
OC 340 POKE 53765,(SO(2,1)+S
        (2,A))
OH 350 POKE 53767,(SO(3,1)+S
        (3,A))
CA 360 NEXT A:FOR O=1 TO 5:N
        EXT O:POKE 559,34
PM 370 POKE 53761,0:POKE 537
        63,0:POKE 53765,0:POK
        E 53767,0:POKE 752,1
EI 380 POSITION 0,21:? "Pres
        s : SPACE BAR to hear
         again":? "
        {8 SPACES}RETURN to l
        oad another"
DI 390 GET #1,K:IF K<>32 AND
         K<>155 THEN 390
NK 400 IF K=32 THEN 290
EA 410 IF K=155 THEN GOTO 10
        0
```

Program 3: Sound Program Writer

```
EO 10 GRAPHICS 0:POKE 710,0:
        POKE 82,0:POKE 752,1:O
        PEN #1,4,0,"K:"
BB 20 POSITION 0,0:? "
        {8 SPACES}SOUND DEVELO
        PMENT SYSTEM
        {8 SPACES}"
DD 30 POSITION 10,1:? "SOUND
        PROGRAM MAKER":GOSUB
        1040
AH 40 ? :? :? "This program
        takes any arrangement
        of{3 SPACES}envelopes
        made with the Sound Ed
        itor"
IE 50 ? "and writes an ENTER
        -able BASIC program  t
        o play them. All you n
```

```
     eed to provide"
KM 60 ? "is the starting lin
     e number of the
     {6 SPACES}prograe and
     a saved sound envelope
     file."
KJ 70 POSITION 13,20:? "PRESS
     ANY KEY"
EI 80 POKE 764,255:GET #1,K:
     ? CHR$(125)
FK 100 ? :? :? "Enter name f
     or load file."
OM 110 ? "{Q}{25 R}{E}"
IE 120 ? "!Dn:filename.  Ext
     ender is!"
PH 130 ? "!autoeatically app
     ended.  !"
PH 140 ? "{Z}{25 R}{C}"
HK 150 POKE 752,0:POSITION 1
     2,14:INPUT FL$:IF FL$
     ="" THEN 90
FF 160 IF FL$(1,1)<>"D" THEN
     90
BK 170 POKE 82,0:? :TRAP 190
     :FOR D=1 TO 15:IF FL$
     (D,D)="." THEN POP :G
     OTO 90
BM 180 NEXT D
HH 190 FL$(LEN(FL$)+1)=".SND
     "
MF 200 CLOSE #2:TRAP 210:OPE
     N #2,4,0,FL$:CLOSE #2
KC 210 TRAP 40000:IF PEEK(19
     5)<>170 THEN 230
II 220 ? :? FL$;" does not s
     eem to exist...":POKE
     752,1:POSITION 0,20:
     ? "{6 SPACES}PRESS AN
FE 230 ? "Okay, loading ";FL
     $;"."
IP 240 CLOSE #2:OPEN #2,4,0,
     FL$:GET #2,BYTE
AC 250 FOR X=0 TO 3:FOR Y=0
     TO 1:GET #2,Z:SD(X,Y)
     =Z:NEXT Y:NEXT X
PH 260 FOR X=0 TO 3:FOR Y=1
     TO 35:GET #2,Z:S(X,Y)
     =Z:NEXT Y:NEXT X
EH 270 POKE 752,1:? CHR$(125
     ):? :? "The next e
     nu is a listing of d
     ifferent voice arrang
     ements. Choose one,"
BM 280 ? "enter the naee of
     the prograe to write,
      and then select the
     starting line"
MM 290 ? "number; the coeput
     er will start writing
     "
PL 300 ? "the prograe, which
      should only take a
     {3 SPACES}few einutes
     .":POSITION 13,20
OM 310 ? "PRESS ANY KEY":POK
     E 764,255:GET #1,K
LO 320 ? CHR$(125)
PD 340 POSITION 12,2:? "ARRA
     NGEMENT MENU"
BB 350 POSITION 10,4:? "-- V
     oice Nuebers --"
HF 360 POSITION 6,6:? "A - 0
     ,1,2,3{9 SPACES}B - 0
     ,1,2"
LP 370 POSITION 6,7:? "C - 0
     ,1,3{11 SPACES}D - 0,1
     "
MH 380 POSITION 6,8:? "E - 0
     ,2,3{11 SPACES}F - 0,2
     "
MP 390 POSITION 6,9:? "G - 0
     ,3{13 SPACES}H - 1,2,3
     "
JF 400 POSITION 6,10:? "I -
     1,2{13 SPACES}J - 1,3"
OM 410 POSITION 6,11:? "K -
     2,3{13 SPACES}L - 0"
OF 420 POSITION 6,12:? "M -
     1{15 SPACES}N - 2"
OO 430 POSITION 6,13:? "O -
     3"
OI 440 POSITION 9,16:? "Your
      choice (A-P) : ?"
ME 450 GET #1,K:IF K<65 OR K
     >79 THEN POSITION 9,1
     6:? "YOUR CHOICE (A-P
     ):   ":GOTO 440
AP 460 V=K-64
DI 480 ? CHR$(125):? :? "Ent
     er naee for the progr
     ae file."
OA 490 POKE 752,0:POSITION 1
     2,14:? "{DEL LINE}";:
     INPUT FLL$:IF FLL$=""
      THEN 480
MC 500 IF FLL$(1,1)<>"D" THE
     N 480
KL 520 ? CHR$(125):? :? "Ent
     er the starting line
     nueber":? "and interv
     al for ";FLL$
DI 530 ? " Enter START,INTER
     VAL"
JB 540 TRAP 540:POSITION 0,1
     2:INPUT N1,N2:TRAP 40
     000
JH 550 ? "OK...":? "--COMPAC
     TING--"
EP 560 LN=35:FOR A=35 TO 1 S
     TEP -1
JM 570 S=0:FOR B=1 TO 4:IF G
     (V,B)=0 THEN GOTO 590
FM 580 S=S+S(B-1,A)
AF 590 NEXT B:NUM=N1
EL 600 IF S<>0 THEN POP :GOT
     O 620
BJ 610 LN=A:NEXT A
AM 620 ? "--- WRITING ---"
BB 630 CLOSE #1:OPEN #1,8,0,
     FLL$
KG 640 D$=STR$(N1):D$(LEN(D$
     )+1)=" LN=":D$(LEN(D$
     )+1)=STR$(LN*2):? #1;
     D$:D$="":N1=N1+N2
LO 650 D$=STR$(N1):D$(LEN(D$
     )+1)=" POKE 53775,3:P
     OKE 53768,0":? #1;D$:
     D$="":N1=N1+N2
HA 660 D$=STR$(N1):D$(LEN(D$
     )+1)=" DIM "
KH 670 FOR A=1 TO 4:IF G(V,A
     )<>1 THEN 690
OF 680 D$(LEN(D$)+1)="E":D$(
     LEN(D$)+1)=STR$(A-1):
     D$(LEN(D$)+1)="$(LN),
     "
KL 690 NEXT A:IF D$(LEN(D$),
     LEN(D$))="," THEN D$(
     LEN(D$))=""
BA 700 ? #1;D$:N1=N1+N2:D$="
     "
PK 710 FOR A=0 TO 3:IF G(V,A
     +1)<>1 THEN 760
FB 720 D$=STR$(N1):D$(LEN(D$
     )+1)=" E":D$(LEN(D$)+
     1)=STR$(A):D$(LEN(D$)
     +1)="$=":D$(LEN(D$)+1
     )=CHR$(34)
MP 730 FOR B=1 TO LN:R=S(A,B
     ):IF R<10 THEN D$(LEN
     (D$)+1)="0"
MM 740 D$(LEN(D$)+1)=STR$(R)
     :NEXT B:D$(LEN(D$)+1)
     =CHR$(34)
BF 750 ? #1;D$:D$="":N1=N1+N
     2
BN 760 NEXT A
HJ 770 D$=STR$(N1):D$(LEN(D$
     )+1)=" POKE 53768,":D
     $(LEN(D$)+1)=STR$(BYT
     E):? #1;D$:N1=N1+N2:D
     $=""
DB 780 D$=STR$(N1)
PP 790 FOR A=0 TO 3:IF G(V,A
     +1)<>1 THEN 820
PP 800 D$(LEN(D$)+1)=" POKE
     ":D$(LEN(D$)+1)=STR$(
     53760+2*A):D$(LEN(D$)
     +1)=",":D$(LEN(D$)+1)
     =STR$(SD(A,0))
AB 810 D$(LEN(D$)+1)=":"
LE 820 NEXT A:IF D$(LEN(D$),
     LEN(D$))=":" THEN D$(
     LEN(D$))=""
BE 830 ? #1;D$:N1=N1+N2:D$="
     "
CD 840 D$=STR$(N1)
CB 850 D$(LEN(D$)+1)=" POKE
     559,0:FOR A=1 TO ":D$
     (LEN(D$)+1)=STR$(LN):
     D$(LEN(D$)+1)=" STEP
     2":? #1;D$:D$="":N1=N
     1+N2
PN 860 FOR A=0 TO 3:IF G(V,A
     +1)<>1 THEN 910
OB 870 D$=STR$(N1)
BE 880 D$(LEN(D$)+1)=" POKE
     ":D$(LEN(D$)+1)=STR$(
     53761+2*A):D$(LEN(D$)
     +1)=",VAL(E":D$(LEN(D
     $)+1)=STR$(A)
HE 890 D$(LEN(D$)+1)="$(A*2-
     1,A*2))+":D$(LEN(D$)+
     1)=STR$(SD(A,1))
OP 900 ? #1;D$:N1=N1+N2
BK 910 NEXT A
BH 920 D$=STR$(N1):D$(LEN(D$
     )+1)=" NEXT A:POKE 55
     9,34":? #1;D$:D$="":N
     1=N1+N2
OH 930 D$=STR$(N1):D$(LEN(D$
     )+1)=" POKE 53761,0:P
     OKE 53763,0:POKE 5376
     5,0:POKE 53767,0:POKE
      53768,0:END":? #1;D$
PK 940 D$="":? #1;D$:CLOSE #
     1
HA 950 ? CHR$(125):? :? "---
     > FINISHED":POKE 752,
     1
BE 960 ? :? :? "Press R to r
     un the prograe again"
KJ 970 ? "{3 SPACES}or Q to
     quit."
KL 990 OPEN #1,4,0,"K:":POKE
      764,255
PM 1000 GET #1,K:IF K<>ASC("
      R") AND K<>ASC("Q")
      THEN 1000
LL 1010 IF K=ASC("R") THEN R
      UN
PN 1020 POKE 752,0:END
CF 1040 DIM FL$(20),G(14,4),
      FLL$(20),S(3,35),D$(
      120),SD(3,1)
MM 1050 RESTORE 1060:FOR X=1
       TO 14:FOR Y=1 TO 4:
      READ D:G(X,Y)=D:NEXT
       Y:NEXT X
IO 1060 DATA 1,1,1,1,1,1,1,0
      ,1,1,0,1,1,1,0,0,1,0
      ,1,1,1,0,1,0,1,0,0,1
      ,0,1,1,1,0,1,1,0,0,1
      ,0,1,0,0,1,1,1,0,0,0
      ,0,1,0,0
JI 1070 DATA 0,0,1,0,0,0,0,1
KJ 1080 RETURN
```

©

IBM Keyboard Customizer

David Engebretsen

This tutorial for the IBM PC/PCjr explains how to customize your computer's keyboard with simple DOS 2.0 or higher commands. Besides reassigning key definitions to your own personal taste, you can create keyboard macros, define as many as 40 function keys and choose new screen modes and color combinations.

Have you ever wished you could change a key on the keyboard into another key, or make a single keystroke spell out an entire phrase? Would you like to move the colon and the semicolon keys, or put the Return key in a more convenient position? It is quite possible do to all this and more with a few simple commands from DOS. You can even create up to 40 different function keys which can be used to print a variety of command words or phrases of any length. With a little more work, you can even change your standard QWERTY keyboard into the efficient Dvorak format which can improve typing speed dramatically.

All this is made possible with the extended screen and keyboard control offered by DOS 2.0 and higher. Though the DOS manual devotes only one page to this subject, the process is not complicated. Let's look first at how to switch key assignments. Then we'll explore how to perform related tasks such as setting the screen mode, changing the background and foreground colors, and positioning the cursor.

Boot Up With CONFIG.SYS

It is suprisingly easy to reassign any key or keys to a location that suits your own personal needs. The first step is to create a CONFIG.SYS file that installs an extended screen and keyboard control device driver when you boot the system. This can be done with the EDLIN system editor included on your DOS disk. From the DOS command prompt (>) simply type EDLIN CONFIG.SYS and press Return. The drive will whir as it opens a new file; after a few moments your screen should look something like this:

```
New file
*
```

Type the following lines, pressing Return at the end of each line:

```
1i
DEVICE=ANSI.SYS
```

Press Ctrl-Break, then type the number 3 and press Return. At this point you have created a configuration file that runs automatically whenever you boot the computer. The result is that the computer is then made ready to accept some new key assignments.

Reassigning Key Definitions

The next step is to do the actual key switching. Since this can involve some odd character sequences, it's easiest to do this from within a BASIC program that stores the needed data in a text file on disk. Here is a program that demonstrates the technique. We'll use it to create a file that changes the uppercase Q to an uppercase D.

```
DN 10 A$=CHR$(27) + "[" + CHR$(3
      4) + "Q" + CHR$(34) + ";"
      + CHR$(34) + "D" + CHR$(34
      ) + "p"
FN 20 OPEN "KEY.TXT" FOR OUTPUT
      AS #1
DD 30 PRINT #1,A$
GM 40 CLOSE #1
```

Save this program as REASSIGN-.BAS and run it. REASSIGN.BAS creates a text file that contains the following character sequence:

```
ESC ["Q"; "D"p
```

CHR$(27) is the ASCII code for the ESC character; this is the control code which changes the uppercase Q into an uppercase D. To implement this change, insert the disk containing your new CONFIG.SYS file, then reboot by pressing Ctrl-Alt-Del simultaneously. This enables the ANSI device driver which in turn allows the keyboard to be redefined.

After answering the time and date prompts, type TYPE KEY.TXT at the DOS prompt and press Return. This action enters the special control characters into the computer's memory. Now whenever you type an uppercase Q, the system substitutes an uppercase D.

Keyboard Macros

The same technique can be used to create a *keyboard macro*—a key that produces a multicharacter word or phrase with just one keystroke. To illustrate, let's redefine uppercase Q so that it prints the phrase *The*

Phrase whenever it's pressed. Reenter BASIC and load the REASSIGN.BAS program again, then replace line 10 as shown here:

```
EP 1Ø A$=CHR$(27)+"["+CHR$(34)+"
      Q"+CHR$(34)+";"+CHR$(34)+"
      The Phrase"+CHR$(34)+"p"
```

Now run the program again. This creates a text file that contains these characters:

ESC ["Q";"The Phrase"p

Type SYSTEM to go back to DOS, then type the KEY.TXT file again. Now whenever you press Q the computer prints *The Phrase* on the screen.

When creating the KEY.TXT file, it is also acceptable to substitute an ASCII code for the character. For example, say that you want to change uppercase Q back to uppercase D. Go back to BASIC again and change line 10 as shown here. Note that instead of D we are using 68, the ASCII code for D:

```
PM 1Ø A$=CHR$(27)+"[81;68p"
```

Run the program, enter DOS, and TYPE the program. Uppercase Q should again produce a D.

40 Function Keys

Now let's create some extra function keys. By supplying an extended ASCII code, you can redefine the ten function keys alone or in conjunction with the Ctrl, Shift, or Alt keys. That comes to four sets of ten, or 40 keys. Rerun the example program after changing line 10 as shown here:

```
A6 1Ø A$=CHR$(27)+"[Ø;84;"+CHR$(
      34)+"DIR"+CHR$(34)+";13p"
```

The following text file is created:

ESC [0;84;"DIR";p

The 0 before the 84 tells the computer to look for an *extended keycode*—a code that signals a special key combination. The extended keycode 84 represents Shift-F1. What we've done is redefine this key combination so that it prints DIR followed by a carriage return. Run the program, exit to DOS, and type KEY.TXT again. Hold down Shift and press F1: the disk directory is displayed.

Note the number 13 just before the *p* in this character sequence. This is the ASCII code for Return. Adding this character to the end of a character sequence has the same effect as pressing RETURN manually on the keyboard. The computer types the letters D-I-R, then issues a Return to carry out the command. You can find a complete list of all the extended keycodes on page G-7 of the IBM BASIC manual.

Screen Modes And Colors

Using a similar method, you can also change the screen color or shift to a different screen resolution. To change colors, replace the lowercase *p* in line 10 with a lowercase *m*, and supply an appropriate color number. For instance, change line 10 as shown here and create a new KEY.TXT file:

```
EN 1Ø A$=CHR$(27)+"[37;44m"
```

Now the program creates this text file:

ESC [37;44m

When this file is TYPEd from DOS the screen turns blue.

The same procedure works for changing the screen mode. Change line 10 to this (note that an *h* is substituted for the *m* in the preceding example):

```
B6 1Ø A$=CHR$(27)+"[=1h"
```

When you TYPE the resulting file from DOS, the screen goes into 40 × 25 color text mode. To obtain 320 x 200 color graphics mode, simply change the number 1 in line 10 to a 4. Pages 13-9 and 13-10 in the DOS 2.0 manual contain a complete listing of all the numbers for different screen modes and color combinations.

The customizations you create using these techniques will stay in effect as long as you are working in DOS or a DOS-related program such as DEBUG or EDLIN. If you're tired of the normal white-on-black screen display, this simple technique can bring a welcome change. Note, however, that these changes disappear if you reboot the computer, go to BASIC, or run an application that imposes its own definitions on the system. ©

Advanced Programming On The Atari ST

Writing sophisticated programs on the Atari ST requires a more thorough understanding of the computer's operating system than was necessary on earlier machines. This article is an introduction to the various operating system routines available to ST programmers. It is an excerpt from COMPUTE!'s ST Programmers Guide *(by the editors of* COMPUTE! Publications.*)*

It seems quite natural to move the mouse pointer to an icon, click on it, and then double-click to open the window. Often you can choose options from menus at the top of the screen by simply pointing and clicking. Click on the menu bar and you can move the window around. You can resize it, expand it to fill the screen, or make the window go away, all with a few clicks of the mouse.

Making things simple for the user requires a wide variety of fairly complex routines for handling input, output, graphics, and so on. These routines are invisible to the user, who simply moves the mouse to point and click. But as a programmer, you may find it helpful to gain some acquaintance with the built-in TOS and GEM routines. The more you know about how they work, the more control you will have over the machine and the more power you can put into programs.

You can call many of these routines in BASIC with the GEMSYS or VDISYS commands, but to get the maximum speed and power from your ST, you'll need either a C compiler or a machine language assembler.

Alphabet Soup

We'll be referring to the various collections of routines by their initials: TOS, VDI, AES, and so on. We'll first look at what they are and what they do.

The Operating System (TOS) is either built into your ST computer in ROM or on a TOS boot disk. The most identifiable element of this operating system is the *Graphics Environment Manager* (GEM) desktop. However, the operation of TOS involves the interplay of many different, specialized components. The first step toward understanding GEM and TOS is to learn the name and function of each of the parts.

The desktop environment is actually a special type of GEM application, which exists only to perform file operations, including running other GEM applications. Every function that it performs, from examining disk directories to running other applications, can be duplicated by any GEM application using the facilities of the AES, VDI, and GEMDOS.

The GEM *Virtual Device Interface* (VDI) provides low-level graphics display and mouse input routines. This routine library includes primitive drawing operations like line, marker, circle, and polygon, as well as display management routines like clipping and block image copying.

The GEM *Application Environment Services,* AES for short, performs higher-level graphic and data management operations for maintaining the GEM desktop environment. Built on top of the GEMDOS file system and the GEM VDI, the AES routines make it much easier for programs to perform mouse and window operations.

The Disk Operating System GEMDOS performs character-oriented file and device input/output (I/O). Many of its routines are used by the GEM AES. GEM applications can also use GEMDOS for file access and for device operations. For the actual low-level operations, GEMDOS calls the *Basic Input/Output System* (BIOS), a group of routines which perform machine-specific tasks on the ST. The Atari XBIOS provides additional machine-specific operations. They are not used by GEMDOS, but are available to applications which need routines not available through GEMDOS or the GEM BIOS.

AES In Detail

When you double-click on an application's icon, the desktop starts executing that application. Although you can move or resize an application's window, its output appears only in that window. An application can redraw its window when it is partially covered by an accessory without overwriting the accessory's window. When you click on the box at the upper left of an application's window, the application stops executing and the desktop returns.

All of these operations would be much harder to perform without the support of the AES, which is composed of a process dispatcher, a screen manager, a desk accessory buffer, and 11 subroutine libraries.

The process dispatcher allows one application and several accessories to wait for a user action simultaneously, a limited form of multitasking. When the system is booted, the accessory buffer is loaded with accessories. The process dispatcher suspends all accessories until a menu item for one of them is selected.

The procedures in the subroutine libraries support common operations on the application environment, the desktop. An application could perform most desktop operations without the AES, using the same VDI routines, but the standard AES routines make the job considerably less difficult. This is a powerful motivation for programmers to make user interfaces more alike, so the programs are easier to learn and use. Nonetheless, nonstandard interfaces can be built where needed, using either VDI or lower-level AES routines.

While the information presented here is not a thorough treatment of the AES, it is intended to be a concise overview of the whole library. It should give you some idea of how the AES is organized, what its constituent parts are and how they interact, and what facilities are available to the GEM programmer. More specific details about each routine's parameters and their values can be found in the Digital Research GEM literature and in sample GEM programs.

Application Library

The application library is, in effect, the gateway to the rest of the AES. Its appl_init routine is used by an application to register with the AES and obtain a process ID. It also tells the AES to set up data structures to keep track of this application or accessory. When the program has finished, the appl_exit routine tells the AES to deallocate the ID number and data structures for this application.

When waiting for user input from the mouse or keyboard, a program can call one of the routines in the event library instead of waiting in a loop. Besides saving the programmer from writing a few more lines of code, a call to the event library freezes the application and allows possible multitasking, if an event for another frozen process occurs first. Calls are available to wait for keyboard, mouse button, message, or timer events. Messages indicating that some action must be performed—like redrawing or moving a window—are usually awaited with an event library call. The routine most commonly used is evnt_multi, which allows an application to wait for more than one event at once.

The menu_bar routine, in the menu library, is used to display or erase the menu bar. Another routine, menu_register, is called by desk accessories to add a name to the Desk menu. Routines are also available to enable and disable menu items, to change the names of menu items, and to display a menu item in reverse video (as a selected item).

Windows

The single most distinguishing feature of the GEM operating environment is the window. The output of applications and accessories running on the ST is displayed on the screen in separate windows, many of which can be moved or changed in size with the mouse. There can be as many as eight windows on the screen at once, and they can overlap in any way, but applications are advised not to use more than four of them if the Desk menu is displayed. Since desk accessories need to have windows available when they are activated, it's best to reserve four of the available windows for accessories. The routines in the window library perform services which are useful in the management of these windows.

The wind_create and wind_delete routines are used to create windows and to dispose of them. When an application calls wind_create to generate a new window, it establishes the maximum possible size of the window and the features with which the window is endowed, such as sliders, a name bar, a full screen box, and so forth. This routine returns a numeric identifier for the new window, called a *handle.* The wind_open call actually displays the window on the screen at a particular location. Conversely, a window can be hidden with the wind_close routine. The expanding and shrinking box effects that are seen when many applications open and close windows are not part of the window library subroutines. Rather, they are independent effects routines from the graphics library, which an application can call for a little more flash.

Several window library routines provide information about windows. Wind_get can provide information about windows, including a window's position and size, and the positions of its vertical and horizontal sliders if it is so equipped. It can also return the handle of the top window on the screen (the window with the highest priority), as well as the set of rectangles that make up the visible portion of a window's work area, which can be a irregular shape if a window is partially covered by another. The wind_set routine is used to change the size and position of a window, the positions of its sliders or its name, or the set of controls attached to the window. It can also move one window to the top of the list of windows.

To determine which window the mouse currently points to, the wind_find routine can be called. Wind_calc doesn't operate on any particular window, but instead determines the work area size of a hypothetical window, given its external size and the set of controls it contains. It can also perform the reverse calculation, determining the external dimensions of a window.

The wind_update routine tells AES that an application is going to draw in a window or that an update is finished. When a window is being updated, no alerts, dialogs, or menus will be displayed in front of the window.

Objects

Most items on the GEM desktop, including menus, alerts, and even windows, are organized as object trees. GEM *objects* include icons, strings, graphic boxes, and editable text fields. *Trees*—linked lists—of these objects can be managed with the routines in the object library. This library includes routines to add and delete objects from a tree, to compare the mouse's position to that of an object, to let the user edit a text object, and to draw the entire

tree on the screen. Object trees are usually stored in a file separate from the application that uses them. These *resource files* can be handled with the resource library, described below.

A *form* is a standard mechanism for getting information from the user. A form usually includes at least one modifiable object, like a text string or a button. In the case of a *dialog*, one type of form, the program needs to call the form_dial routine to indicate that a dialog is beginning, and then draw the object tree which comprises the dialog, using the objc_draw routine from the object library. The form_do routine should then be called to perform the interaction. Another call to form_dial restores the area of the screen where the dialog took place.

Two other forms, the *alert* and the *error box*, are more limited in their content and, accordingly, easier to implement. In the case of an alert, the AES builds an object tree containing the text string passed in the call to form_alert and handles all the details of display and interaction during that one call. The error box is even simpler. All that is needed is the number of a GEMDOS error code. An object tree containing the text which corresponds with that error will be displayed when form_error is called.

Special-Purpose Libraries

The AES graphics library routines are lower-level interfaces to the display screen and mouse. The most commonly used routine in this library is graf_handle, which returns the handle—the identifier—for the currently opened VDI workstation. Every GEM application must call this routine at its start, after calling appl_init, since the handle is needed to open a new VDI virtual workstation to draw in. It is also necessary for calling VDI drawing routines.

To manage the mouse at a low level, the graf_mouse routine can be used to change the shape of the cursor. Graf_mkstate monitors the positions of the mouse, its buttons, and the keyboard, while graf_watchbox modifies the state of a box object depending on whether or not the mouse's pointer is inside or outside the box.

Several common graphic effects are also available through the graphics library. Graf_rubberbox draws a rectangle between a fixed point and the mouse's position, changing the size of the box as the mouse moves. A moving box of fixed size attached to the mouse's position can be animated with graf_dragbox, while graf_movebox just moves a box between two positions without any consideration of the mouse. Graf_growbox and graf_shrinkbox are two animation routines which can be called when opening and closing windows, respectively, to show a box which moves and changes size.

The file selector library contains but one lonely routine, fsel_input. This displays a standard dialog box, called a *file selector,* on the screen and allows the user to choose a filename from the directories of the various disks in the system. When the interaction is complete, it returns the pathname of the selected file to the calling application.

The objects that an application uses—its menus, dialogs, and so forth—can be stored separately from the application's code in a resource file. Resource files containing these objects are created with the *GEM Resource Construction Set* program. The resource library can then be used to load this file and to access its contents.

The rsrc_load routine searches for a resource file with a particular name and attempts to load it into memory. Rsrc_gaddr can be used by the application to find the address of a particular object or tree in the resource file that has been loaded. To allow applications to run with different screen resolutions in different display modes, all the sizes and positions of objects in a resource file can be expressed in characters instead of pixels. When the resource file is loaded, rsrc_obfix must be called to convert the sizes into pixels in the current display mode. When a resource file is no longer needed, rsrc_free deallocates the memory space that the resource occupies.

Using GEMDOS

For many kinds of programs, the windowed GEM environment might not be needed, so an alternative is available. A complete set of character-oriented I/O functions is provided in the GEMDOS library. Unlike many operating systems, GEMDOS is easily called from languages like C. Instead of taking parameters in the microprocessor's internal registers, which are not directly controllable in high-level languages, parameters are passed to these routines on the stack in the same way that parameters are passed between C functions.

Even GEM applications need to call GEMDOS at least occasionally. For instance, the AES scrap facility expects applications to store and read the contents of the Clipboard directly from disk. Any program which handles some kind of document—a spreadsheet, word processor, database, and so on—will also need to call GEMDOS to load and store files.

To call GEMDOS, the 68000 microprocessor's TRAP #1 instruction must be executed. The last word pushed on the stack gives the number of the routine requested. If the routine returns a value, it will be stored in the 68000's D0 register. Although this register cannot be directly read by a C program, C functions also return results in this register. A GEMDOS call can return a value in exactly the same manner as a call to another C function.

Process Functions

00 **Pterm0( ).** Terminate with a return code of 0.

49 **Ptermres(size,code).** Terminate, but keep the program's code in memory. A 32-bit parameter indicates how much memory should remain allocated. A 16-bit parameter gives the return code. This function is used by background programs, like print spoolers, to give up control of the foreground process.

75 **Pexec(runflag,pathname,tail,environ).** Load a program from disk. A one-word parameter indicates whether it should be run or not (00=run, 03=load only). The second parameter is a pointer to the pathname of the file. Parameter 3 points to a command tail for the program, and parameter 4 is a pointer to its environment strings. If the file was loaded only, the result of the function is the load address. If it was executed, the result is its return code.

76 **Pterm(code).** Terminate, returning a one-word return code.

Device I/O Functions

01 **Cconin( ).** Read a character from the console. No parameters are needed.

02 **Cconout(char).** Write a character to the console. A single, word-length parameter contains the character in its low byte.

03 **Cauxin( ).** Read a character from the auxiliary device.

04 **Cauxout(char).** Write a character to the auxiliary device.

05 **Cprnout(char).** Write a character to the printer.

06 **Crawio(char).** If the parameter is not $00FF, write it as a character to the console; otherwise, return a character from the console with no echo, including control characters.

07 **Crawcin( ).** Read a character from the console with no echo and no control character trapping.

08 **Cnecin( ).** Read a character from the console with no echo, but trap ^C, ^S, and ^Q.

09 **Cconws(string).** Write a zero-terminated string to the console. The long word parameter contains the address of the string.

10 **Cconrs(buffer).** Input a line of characters from the console, allowing line editing. The long word parameter contains the address of a buffer, the first byte of which holds the buffer's length. The second byte of the buffer will get the length of the string, and the string will be zero-terminated.

11 **Cconis( ).** Check the status of the console input device. Returns −1 if a character is waiting, 0 if none is available.

16 **Cconos( ).** Check the status of the console output device. Returns −1 if it is ready to receive a character, 0 if it is not ready.

17 **Cprnos( ).** Check the status of the print device.

18 **Cauxis( ).** Check the status of the auxiliary input device.

19 **Cauxos( ).** Check the status of the auxiliary output device.

Time Functions

42 Tgetdate( ). Return the current system date. Bits 0–4 of the result contain the date, 5–8 contain the month, 9–15 contain the year minus 1980 (up to 2099).

43 Tsetdate(date). Set the current system date to the word value in the parameter.

44 Tgettime( ). Return the current system time. Bits 0–4 contain seconds/2, 5–10 contain minutes, 11–15 contain hours.

45 Tsettime(time). Set the current system time to the word value in the parameter.

System Functions

32 **Super( ).** Enter 68000 supervisor mode.

48 **Sversion( ).** Return the GEM version number.

Drive Functions

14 **Dsetdrv(drive).** Set the default disk drive to the value passed as a parameter. Values 0–15 indicate drives A–P.

25 **Dgetdrv( ).** Return the value of the current default drive.

54 **Dfree(buffer,drive).** Ask for information about a disk. The first parameter contains the address of buffer to receive the information. The second parameter is a 16-bit value indicating the drive being queried. The buffer gets four values: free space, number of clusters on drive, size of sector in bytes, size of cluster in sectors.

57 **Dcreate(pathname).** Create a subdirectory. The long word parameter contains the address of null-terminated string for the pathname of the new directory. If the result is nonzero, the operation failed.

58 **Ddelete(pathname).** Delete a subdirectory.

59 **Dsetpath(pathname).** Set the current default pathname to the string addressed by the parameter.

71 **Dgetpath(buffer,drive).** Store the current directory for a drive in a 64-byte buffer addressed by the first parameter. The second parameter indicates which drive (00=default drive, 01–10=A–P).

File Functions

26 **Fsetdta(DTAbuffer).** Use the long word parameter to set the address for a 44-byte file data buffer. This buffer is used only when searching for a file (routines 78 and 79).

47 **Fgetdta( ).** Return the address of the file data buffer.

60 **Fcreate(pathname,attributes).** Create a file named by the string addressed by the first parameter. An additional 16-bit parameter indicates the file's attributes (01=RO, 02=hidden). The result is a 16-bit file handle.

61 **Fopen(pathname,access).** Open the file named by the string addressed by the first parameter. An additional 16-bit parameter indicates type of access (00=read only, 01=write only, 02=read and write). The handle of the file is returned as the function's result.

62 **Fclose(handle).** Close a file. The handle of the file is passed as a parameter.

63 **Fread(handle,count,buffer).** Read from a file. The first parameter is the handle of an open file and the second is a four-byte count for the transfer. The third parameter is the address of a buffer to which the bytes are to be read. The result returned by the function is the number of bytes.

64 **Fwrite(handle,count,buffer).** Write to a file. The function uses the same parameters as Fread.

65 **Fdelete(pathname).** Delete the file named by the string pointed to by the parameter.

66 **Fseek(count,handle,operation).** Move the file pointer. The first parameter is a signed long word representing a byte count. The second parameter is a file handle. The third indicates the meaning of the byte count (00=absolute position *N* bytes after the start of file, 01=*N* bytes forward or backward from current position, 02=*N* bytes before the end of the file). As a result, it returns the absolute file pointer position.

67 **Fattrib(pathname,operation,attributes).** Read or change the attributes of file. The first parameter is a pointer to a pathname for the file. The second parameter is 00 for get, 01 for set. The third parameter is a word containing the attributes.

69 **Fdup(handle).** Return a copy of the file handle passed as a parameter.

70 **Fforce(handle1,handle2).** Force the first parameter, a file handle, to point to same device as the second parameter, also a handle.

78 **Fsfirst(pathname,attributes).** Search for the first file which matches the search string addressed by parameter 1. The string can contain the * and ? wildcards. Parameter 2 contains the attribute flags of the file. The 44-byte file data buffer set by Fsetdta holds size of file in bytes 26–29, and the file's name and type in bytes 30–43.

79 **Fsnext( ).** Search for another match to the file, using the data buffer at DTA. This function takes no parameters.

86 **Frename(0,oldname,newname).** Rename a file. Parameter 1 is a word with the value 0; parameter 2 is a pointer to the old pathname of the file; parameter 3 points to the new pathname.

87 **Fdatime(buffer,handle,operation).** Get or set a file's date and time information. Parameter 1 is a pointer to a two-word buffer—a time word and a date word. Parameter 2 is a file handle, and parameter 3 is a word which indicates which operation to perform (00=set, 01=get).

Memory Functions

72 **Malloc(count).** Allocate some number of bytes to the calling application. The length of the block requested is passed in the parameter, a long word. It returns a value of 0 if the request fails, or the address of the block allocated if it succeeds. If the parameter has a value of −1, the number of free bytes is returned instead.

73 **Mfree(address).** Free a block of memory. The four-byte parameter should contain the address of the block.

74 **Mshrink(0,address,length).** Reduce the size of an allocated block of memory. The first parameter must be a word with value 0, the second parameter is the address of the block, and the third parameter is the number of bytes which should remain in the block. @

Block PEEK And POKE For Atari

Ronald R. Lambert

Here is a convenient way to eliminate long initialization delays caused by POKEing large amounts of data into memory. It works entirely in BASIC and works very fast. The demonstration program moves the entire character set in an instant and redefines the keyboard as a Dvorak layout. This technique can be used on all Atari 400/800, XL, and XE computers and is recommended for intermediate to advanced BASIC programmers.

PEEK and POKE are among the fastest commands in BASIC. But because they handle only one byte at a time, it can take a while to transfer large blocks of data from one area of memory to another. We've all waited while programs with long loops PEEK a series of memory locations, or READ numbers from DATA statements, and then POKE the numbers into memory somewhere else. Perhaps the program is redefining the character set, or setting up player/missile graphics, or building a machine language subroutine, or creating a new keyboard definition table. Whatever's happening, it slows things down.

Lengthy FOR-NEXT loops with PEEK and POKE or READ and POKE are the primary cause for tedious delays while these programs initialize. No one likes to sit staring at a blank screen for very long. The program usually prints a message like "Please wait while I initialize," but isn't there a better way? Sometimes a machine language subroutine can help speed things up, but if you can't write in ML yourself, finding a routine exactly suited to the needs of your program can be difficult.

Fortunately, Atari BASIC's flexibility provides a solution. It's possible to transfer large blocks of data from Read Only Memory (ROM) or program lines to any area of Random Access Memory (RAM) virtually instantaneously—with BASIC commands only. The secret is called the *string offset* technique. By modifying the variable value table (a section of memory which keeps track of BASIC program variables), you can redefine any string and relocate it anywhere in memory.

Here's a quick overview of how the technique works. Suppose you set up a string called ROM$ which contains a block of data found in ROM—the character set data, for instance. Next, you set up another string called RAM$ in the area of RAM to which you want to move the data contained in ROM$. To copy the data from ROM to RAM, then, all that's required is the simple statement RAM$=ROM$. Is that easy enough? Using the string offset technique, any portion of ROM—or all of ROM, if you make the strings big enough—can be copied into RAM in the blink of an eye.

The Variable Value Table

To use the string offset technique, you have to learn how to modify the variable value table and the string offset pointers. This isn't too difficult if you tackle the job one step at a time.

The first step is to make things easier for ourselves by insuring that ROM$ and RAM$ are the first variables found in the table. We can do this by making ROM$ and RAM$ the very first variables defined in the program. Enter NEW as a direct command before typing the first program line containing these names. Then dimension the variables in this order:

10 DIM ROM$(*length*),RAM$(*length*)

where *length* is the length of each string in characters as required by your program. If you're moving character set data, for instance, you'd dimension these strings to the number of bytes in the character set—1024 bytes in graphics mode 0 or 512 bytes in modes 1 and 2. (Usually you'll dimension ROM$ and RAM$ to the same length if you're transferring a block of memory.)

However, if you are using an Atari 400 or 800 with the old BASIC revision A, a major caveat applies. *You cannot move blocks of memory that are exact multiples of 256 bytes.* Attempting to move blocks of this size will trigger the infamous BASIC lockup bug, freezing your

computer until you turn the power off and back on—which will, of course, result in the loss of your program. *[For more information on the lockup bug, see this month's "Readers' Feedback" column. -Ed]* You can determine your version of BASIC by entering PRINT PEEK (43234). If the value returned is 162, you have revision A. If 96 is returned, you have revision B (built into most 600XL and 800XL models), and 234 indicates revision C, available on cartridge from Atari and built into the XE models.

The string offset technique will work as described with revision A as long as you make sure your block length is not an exact multiple of 256. So for this example you should substitute 1025 instead of 1024. This will transfer an extra byte of memory following the character set, but that doesn't cause any problems and it prevents the lockup bug from biting.

The second step is to make BASIC think that ROM$ is actually 1024 bytes long (remember, use 1025 for revision A BASIC). The DIM statement reserves memory for the string but doesn't actually define the string. Use a line like this:

```
20 ROM$(length)=" "
```

By defining the last character in the string as a space, BASIC is forced to treat ROM$ as a 1024-character-long string, even though no other characters have been defined.

The third step is to calculate the location of the variable value table in memory, with a statement like this:

```
30 VT=PEEK(134)+PEEK(135)*256
```

The variable VT equals the starting location of the variable value table. Each string which is declared in an Atari BASIC program has eight bytes in this table. We'll see the significance of these bytes in a moment.

After these variables are set up, the first eight bytes in the variable value table (VT to VT+7) contain information for ROM$, and the next eight bytes (VT+8 to VT+15) contain information for RAM$.

Locating The Strings

To use the string offset technique, we're primarily interested in the third and fourth bytes for each of these two variables in the variable value table. The memory locations for these bytes can be expressed as VT+2 and VT+3 for ROM$, and VT+10 and VT+11 for RAM$.

What do these bytes signify? Briefly, each pair of bytes is a low-byte/high-byte combination that indicates the *relative displacement* of each string from the starting location of the first string in the program. Since we've made sure that the first string in the program is ROM$, the values stored in VT+2 and VT+3 for ROM$ will both be zero. And since we've also made sure that RAM$ is the second string in the program, the values stored in VT+10 and VT+11 for RAM$ depend on the length of ROM$.

For instance, if ROM$ is dimensioned to 1024, then the memory which BASIC sets aside for RAM$ must begin 1024 bytes after the start of ROM$ to leave room for ROM$. Therefore, the value stored in VT+10 is zero, and the value stored in VT+11 is four. (Since VT+11 is the high byte of the offset, it's multiplied by 256, which equals 1024.)

Actually, the memory for ROM$ and RAM$ does not begin at these locations. Instead, you have to add another value indicated by the *string offset pointers* at memory locations 140 and 141. If you use this statement:

```
40 SF=PEEK(140)+PEEK(141)*256
```

then the variable SF returns the number that should be added to the relative displacement values given in the variable value table. (Since the relative displacement of the first string is zero, this means that SF always equals the address of the first string.)

The reason for this seemingly complicated arrangement, incidentally, is that the computer can now easily relocate strings as the program length changes simply by altering the offset pointers.

Setting The Table

Now it's clear how the string offset technique works: We can relocate a string anywhere in memory by merely POKEing different values into its relative displacement indicators in the variable value table.

For example, suppose we want to move ROM$ to the starting memory address of the standard character set in ROM, which is location 57344. We subtract the amount of the string offset (SF) from 57344, and convert the remainder into low-byte/high-byte numbers. Then all we have to do is POKE LS into VT+2 and POKE HS into VT+3, and ROM$ is moved to the proper location. The statements might look like this:

```
50 S=57344-SF:HS=INT(S/256):
   LS=S-HS*256
60 POKE VT+2,LS:POKE VT+3,HS
```

Now we can turn our attention to RAM$.

The usual place to set up a new character set is below RAMTOP—the memory location returned by PEEK(106)*256. Some people prefer to move RAMTOP down by POKEing a lower number into register 106, issue a new GRAPHICS command to set up a new display list and screen memory below the altered RAMTOP, and then put the new character set above the new RAMTOP. There are advantages and disadvantages to each method, including a "RAMTOP dragon" to watch out for. We'll stick to the easiest method for this example. Let's simply put the new character set eight pages (2048 bytes) below RAMTOP. This leaves enough room for the 1024-byte character set, plus another 1024 bytes for the display list and screen memory in graphics mode 0.

We move RAM$ to this location by figuring the proper values and POKEing them into VT+10 and VT+11:

```
70 RAMPAGE=PEEK(106)-8
80 S=RAMPAGE*256-SF:HS=INT
   (S/256):LS=S-HS*256
90 POKE VT+10,LS:POKE VT+11,HS
```

Finally, all that remains is one simple step:

```
100 RAM$=ROM$
```

Instantly, the character set in ROM is copied into RAM, where it can be customized to suit our purposes.

Two Potential Problems

There are two things to watch out for when using the string offset technique. First, if you set up a string in a section of RAM where vital tables or pointers are stored,

then do anything to change the contents of the string, or press BREAK and enter a direct command (which causes BASIC to shift strings and all their contents in memory), the computer may behave very strangely. You'll probably have to turn the machine off and on again to regain control.

Second, you cannot POKE a negative number into the variable value table without getting an error message. How, then, can you move a string to a location in memory lower than the value (SF) indicated by the offset pointers? Simple. POKE the offset pointers to zero, and POKE memory locations 140 and 141 to zero. (Make sure you do this before relocating any strings, or they'll all be moved again when you change the offset pointers.) *But don't leave zeros in locations 140 and 141.* Save the original values and POKE them back in when you're finished transferring the data.

A Dvorak Demo

The program following this article demonstrates the string offset technique and accomplishes several things. First, it copies the standard character set from ROM into RAM (eight pages below RAMTOP) and modifies it so that the CTRL key characters can be recognized more easily. If you press CTRL-A, for example, you won't get the usual graphics symbol; you'll get an underlined A, so you can see at a glance which keys to press to type that character. This way, you can enter ATASCII (Atari ASCII) characters directly into memory from statements in program lines without using DATA statements and slow, one-byte-at-a-time POKEs, because all the characters are immediately recognizable. This character set modification is accomplished in an eye-blink.

Second, the program copies the keyboard definition table from ROM and loads it into memory page 6, a normally unused portion of RAM from locations 1536–1791. Then the program modifies the keyboard table to create a Dvorak keyboard layout. Designed by August Dvorak after 20 years of scientific study and testing, the Dvorak keyboard makes things as easy as possible for typists, in contrast to the conventional QWERTY keyboard, which doesn't put the most frequently used keys on the home row. Many typists are able to convert from QWERTY to Dvorak touch-typing within a few days, and often find they can type faster with substantially fewer errors.

The Dvorak keyboard portion of the program will not work with the older 400 and 800 models because it relies on the KEYDEF pointer at locations 121–122. This pointer was added to the improved operating system in the XL and XE models, and is not implemented in the original Atari operating system ROMs. Owners of 400s and 800s can still use the redefined character set portion of the example by simply omitting all lines numbered higher than 215. If you are using revision A BASIC, you'll also need to change the 1024s in lines 10 and 20 to 1025s.

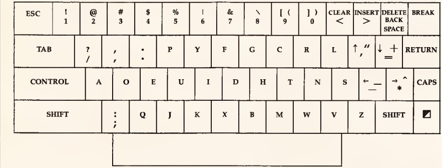
Notice that this program must deal with the problem mentioned above: The memory address 1536 is lower than the value for SF, so the string offset pointers at locations 140 and 141 have to be changed.

A FOR-NEXT loop is used to enter ATASCII characters 0 through 26, so this part of the program takes a little longer—almost a whole second. You could make it run even faster by typing the CTRL key characters directly in string assignment statements, as seen in lines 140 to 170. This is where the new character set could come in handy.

As a final bonus, the program demonstrates a customized keyboard entry routine that works faster than the GET function. It does this by reading a hardware location (53769), then using the keyboard conversion table located in ROM (64337 to 64592) to translate the keyboard codes into ATASCII codes the same way the operating system does.

When the program runs, it lets you toggle back and forth from QWERTY to Dvorak, just like on an Apple IIc. Press SHIFT-ESC to

Atari Dvorak Keyboard Layout



(800 XL keyboard shown; others may vary slightly)

toggle. If you become a real Dvorak fan, you can even find keycap stickers at many office supply stores to modify your keyboard. The accompanying figure shows the Dvorak layout.

Additional Notes

A few modifications to the standard Dvorak layout were necessary because of the special functions and extra keys on the Atari keyboard. The seldom-used brackets may be typed with CTRL-9 or CTRL-0. The += key, normally located at the upper right of the Dvorak keyboard, has been moved down. The *\ key has been retained in its standard Atari position because these characters have extra use as arithmetic functions in programming. Since the Atari has no cent symbol, this has been replaced with the vertical line as uppercase 6. In place of the asterisk (uppercase 8 on the Dvorak keyboard) is the backslash. The ' " key has been exchanged with the ;: key to avoid conflict between the CTRL-up arrow and CTRL-semicolon.

The Atari logo key is the inverse video key on the Atari 400/800, and it is reversed with the right SHIFT key on those models. Regrettably, the Atari has no dash in its character set. While the useless underline could be redefined as a dash for screen display, most Atari printers also lack the dash.

If you enter NEW or load a new program after this one is run, the new character set with readable CTRL key characters remains active (as long as the new character set is not overwritten). Press SYSTEM RESET or POKE 756,224 to restore the old character set. The following POKEs switch on the Dvorak keyboard even after a NEW command: POKE 121,0:POKE 122,6. To switch back to QWERTY, use POKE 121,81:POKE 122,251.

The next time you need to transfer large blocks of data from one portion of memory to another, try using the string offset technique. It gives you the best of both worlds: the convenience of BASIC and near-machine language speed. Never again will you have to sit staring at a blank screen waiting for your programs to move large amounts of data in memory.

Dvorak Keyboard Demo

For instructions on entering this listing, please refer to "COMPUTE!'s Guide to Typing In Programs" in this issue of COMPUTE!

```
CM 10  DIM ROM$(1024),RAM$(10
       24):REM These variable
        names must be the fir
       st entered.
HI 20  ROM$(1024)=" ":GOSUB 6
       0:S=57344-SF:GOSUB 70:
       POKE VT+2,LS:POKE VT+3
       ,HS:REM This moves ROM
       $.
BH 30  RAMPAGE=PEEK(106)-8:S=
       RAMPAGE*256-SF:GOSUB 7
       0:POKE VT+10,LS:POKE V
       T+11,HS:REM This moves
        RAM$.
IF 40  RAM$=ROM$:REM That is
       all it takes! ROM data
        is now copied into RA
       M.
PD 50  SETCOLOR 1,0,0:SETCOLO
       R 2,0,10:SETCOLOR 4,8,
       6:GOTO 80
IP 60  VT=PEEK(134)+PEEK(135)
       *256:SF=PEEK(140)+PEEK
       (141)*256:RETURN
KM 70  HS=INT(S/256):LS=S-HS*
       256:RETURN
IA 75  REM Now modify the cha
       racter set:
KM 80  RAM$(513,520)=RAM$(98)
       :RAM$(521,728)=RAM$(26
       6):FOR X=520 TO 728 ST
       EP 8:RAM$(X,X)=CHR$(25
       5):NEXT X
OK 90  RAM$(769,776)=RAM$(114
       ):RAM$(776,776)=CHR$(2
       55):RAM$(985,992)=RAM$
       (218):RAM$(992,992)=CH
       R$(255)
MP 100 POKE 756,RAMPAGE:REM
        Set the CHBAS pointer
        to start of new char
       acter set.
ON 110 ? "New character set
       ready.":? "CONTROL A=
       ";CHR$(1);", CONTROL
       Z=";CHR$(26)
MO 115 REM Press RESET or PO
       KE 756,224 to restore
        old character set.
AB 120 CLR :DIM ROM$(256),RA
       M$(256):ROM$(256)=" "
       :GOSUB 60:LSF=PEEK(14
       0):HSF=PEEK(141):POKE
        140,0:POKE 141,0
DE 125 REM Now copy keyboard
        definition table fro
       m ROM to page 6 of RA
       M:
NH 130 S=64337:GOSUB 70:POKE
        VT+2,LS:POKE VT+3,HS
       :S=1536:GOSUB 70:POKE
        VT+10,LS:POKE VT+11,
       HS:RAM$=ROM$
LL 135 REM Now change keyboa
       rd definition table t
       o Dvorak layout:
IH 140 RAM$="nhs":RAM$(6)="t
       -*r lg":RAM$(14)="c'=
       k j":RAM$(22)="xq;":R
       AM$(33)="w vb mz":RAM
       $(41)="p .f"
BJ 150 RAM$(46)="y,/":RAM$(5
       7)="ude":RAM$(62)="io
       ":RAM$(65)="NHS":RAM$
       (70)="T_^R LG":RAM$(7
       8)="C +K J/ XQ:"
NI 160 POKE 1614,34:RAM$(92)
       ="!":RAM$(95)="@'W VB
       MZ":RAM$(105)="P .F"
       :RAM$(110)="Y,?( )&"
CH 170 RAM$(118)="\":RAM$(12
       1)="UDE":RAM$(126)="I
       O"
FK 180 FOR X=0 TO 26:READ A:
       RAM$(A)=CHR$(X):NEXT
       X:POKE 1687,123:POKE
       1706,96:RAM$(177)="["
       :RAM$(179)="]"
BH 190 DATA 175,192,164,142,
       186,187,172,140,130,1
       90,147,145,139,166
JC 200 DATA 129,191,169,151,
       137,131,134,185,163,1
       61,150,174,167
HA 205 REM Now restore offse
       t pointers and set up
        custom keyboard entr
       y routine:
LL 210 POKE 140,LSF:POKE 141
       ,HSF:CLR :DIM K$(1):Q
       D=64337:LK=9:C=0:I=0:
       POKE 753,0:? "New key
       code table ready.":?
DG 215 REM Press SHIFT ESC t
       o toggle between QWER
       TY and Dvorak keyboar
       d.
CE 220 ON PEEK(753)<>3 GOTO
       220:K=PEEK(53769):IF
       K=39 OR K=60 OR K=92
       THEN GOTO K*10
NG 230 A=PEEK(QD+K):K=K+C*(A
       >96 AND A<123):K$=CHR
       $(PEEK(QD+K)+I):? K$;
       :POKE 753,3*(LK=K):PO
       KE 20,0
JA 240 ON PEEK(753)<3 GOTO 2
       60:IF PEEK(20)<24 THE
       N 240
MN 250 IF PEEK(753)=3 THEN ?
        K$;:GOTO 250
JA 260 LK=K:POKE 764,255:GOT
       O 220
JE 265 REM Type a letter twi
       ce then hold it down
       to start autorepeat.
IL 390 I=128*(I=0):GOTO 610:
       REM Inverse video tog
       gle
FD 600 C=64*(C=0):REM Cap/lo
       wercase toggle
MM 610 POKE 753,0:GOTO 260
EG 920 QD=1536+62801*(QD=153
       6):POKE 712,134+66*(Q
       D=1536):GOTO 610:REM
       Green border=Dvorak.
       Blue=QWERTY.
```

©

Screen Machine II: A Sketchpad With Pull-Down Menus For PC and PCjr
Part 1

Charles Brannon, Program Editor

Pull-down menus in IBM BASIC? It's no fantasy—presented here is a full-featured drawing program that illustrates the convenience and flexibility of a menu-driven user interface. Next month, in Part 2, we'll show how you can add these menu routines to your own programs. This month's drawing program, "Screen Machine II," runs on any IBM PC or compatible with a color/graphics adapter and BASICA, or a PCjr with Cartridge BASIC. Joystick or touch tablet optional.

Software features first popularized by the Apple Macintosh are finding wider acceptance throughout the computer industry. Pull-down menus and point-and-click selections have become a way of life among owners of the Macintosh, Commodore Amiga, and Atari ST. With the advent of operating system veneers such as *Microsoft Windows, GEM,* and *Topview* for IBM machines, even more people are getting excited about mousing around on their computer.

Without the software tools to implement these techniques, though, programmers have to laboriously write all the routines needed for pull-down menus, icon selection, and windowing, taking time away from programming the application itself. Tools such as *Windows* and *GEM* do the trick for advanced programmers, but BASIC programmers have to reinvent the wheel if they want to add these useful features.

You might think BASIC is not fast enough to emulate the features of operating systems written in turbocharged 8088 or 68000 machine language, but it's almost always possible to tease just a little more power out of BASIC. Although a complete mouse-based user interface is a bit much to expect, I've developed a set of generalized subroutines that any BASIC programmer can use to support fancy pull-down menus in Microsoft Advanced BASIC (BASICA) or PCjr Cartridge BASIC. The routines require bit-mapped graphics, so you need a color/graphics adapter if you're using a PC or compatible. (The PCjr has a built-in color/graphics adapter.) By changing only a few small subroutines, the package can be adapted for other graphics cards and pointing devices.

Rather than illustrate these routines with a plain-vanilla demo program, I though I'd provide a more convincing illustration: a full-featured drawing sketchpad. "Screen Machine II" is a descendant of the original "Screen Machine," a drawing program published about two years ago in COMPUTE!'s PC & PCjr Magazine.

The original Screen Machine used a traditional command-driven user interface. Individual keystrokes were required to activate special commands. For example, to draw a line, you first pressed the space bar to "nail down" one endpoint, marking the spot with a cross. You then moved the cursor to a new position and then pressed L to connect the marked spot with the new cursor position. A line was drawn to connect the points. To draw a circle, you first set your mark to represent the center of the circle, then moved the cursor to a point along the desired circumference. You had to visualize the circle

in your mind, because it wasn't actually drawn until you pressed C.

Although Screen Machine had plenty of features (and in the hands of a talented artist was capable of making beautiful pictures), the stumbling block was the indirect method of using the program. You had to memorize every command or frequently refer to a list of commands. This approach works well once you've mastered a program, but it can alienate the newcomer or occasional user. In a drawing program, especially, it's crucial not to break the flow of ideas between the artist and the canvas.

Screen Machine II

Thanks to pull-down menus, you don't have to memorize a lot of commands to have fun with Screen Machine II. All of the functions are available for selection whenever you need them.

The listing following this article is the minimum required to publish Screen Machine II as a ready-to-run program. Screen Machine II needs almost all of BASIC's 64K memory space, so the original program listing with full comments didn't leave enough memory to run. Next month, however, we'll present a fully REMarked version that shows exactly how the program works, along with a tutorial on using the menu routines in your own programs.

When you first run Screen Machine II, there is a short delay, then the drawing screen appears. The top line of the screen shows which menus are available: *Picture*, *Tools*, and *Preferences*. Your color palette appears at the bottom of the screen, initially showing boxes filled with cyan, magenta, and white paint.

A pointer cursor appears near the middle of the screen. You use this pointer to select items from menus, dip into the paint to change your drawing color, or to draw figures. The pointer can be controlled with a joystick, a touch tablet such as the KoalaPad, or the cursor keys (make sure Num Lock is in the correct position for cursor control). When using the joystick, you may want to unlock it for free-floating movement.

If you don't have a joystick or touch tablet, you can disable the joystick routine in Screen Machine II to prevent interference with the cursor controls. Change line 340 from FROZEN=0 to FROZEN=-1.

To use a light pen or mouse controller, you need to modify the subroutine at line 20000, which we'll discuss next month. The use of menus, though, is not tied to the actual pointing device used, such as a mouse.

Calibrate Your Joystick

The pointer responds most naturally with a touch tablet, so the program is initially set up to use a KoalaPad. The KoalaPad simulates the joystick, but has a greater range, so if you use a joystick with Screen Machine II you'll only be able to position the pointer within the upper-left quadrant of the screen. You may also have problems using a different touch tablet, since not all tablets return the same values. When you first run the program, then, you need to *calibrate* your joystick or tablet. The calibration option is available under the *Preferences* menu, but we haven't discussed how to use the menus yet. And until you calibrate your joystick, you probably can't access the menu item that is used to select the calibrate option. Fortunately, you can also press the J key—a keyboard shortcut—to activate the calibration feature.

After you press J, you're first asked to move the joystick to the upper-left corner, then press the button. (Screen Machine II only uses the top button on the joystick, or the left button of the touch tablet.) This first action sets the origin of your pointing device. If you're using a touch tablet, it is vital that at this point you merely lift the pen off the tablet surface and press the button. This lets Screen Machine II know when you are pressing down on the tablet, and when you lift the pen off the tablet. The value for "pen up" is the same as the coordinates for the upper-left position of the tablet, so it's best just to press the button without touching the tablet surface, in order to make sure that Calibrate sees the right value. With a joystick, move the stick to the northwest corner before you press the button.

Next, you're asked to move the joystick to the lower-right corner, then press the button. With the touch tablet, move the stylus or your fingertip to the southeast corner of the tablet, and while pressing *firmly* with the stylus, click the button. It may be best to use a position slightly above and to the left of the lower-right position, since if you put the pen off the tablet surface, no value is generated.

In general, you must press very firmly against the tablet surface, almost to the point of scoring the tablet, in order to avoid false readings caused by intermittent stylus contact. These false readings aren't caused by Screen Machine II, but by the tablet. A special routine could be used to compute the average position of the touch tablet within the last second, then reject values far out of range, but this would slow down the program to a crawl. As it turns out, this jitter is rarely a problem, since the BASIC program samples the touch-tablet too slowly to see many of the transient glitches.

After calibration, you should be able to move the pointer freely as you slide your finger or stylus across the tablet surface, or by moving the joystick. Control may seem clumsy at first, especially with the joystick, but improves considerably with practice. If you get no response at all, check the joystick or tablet cabling, and press J to calibrate again.

Keyboard Control

If you don't have a joystick or touch tablet, you can use the keyboard cursor controls to move the pointer arrow. The keyboard isn't the fastest way to scurry across the screen, but it is exact. The joystick, however, overrides keyboard control (although you can use a properly calibrated touch tablet along with the keyboard), so you need to press the K key right away to freeze the joystick and enable keyboard control. The K key alternately freezes and reenables the joystick, and is a keyboard shortcut for the *Keyboard* command on the *Preferences* menu.

There are two ways to use the keyboard controls. The pointer can move one pixel at a time for fine movements, but it could take all day to inch your way across the

screen. If you press a cursor key rapidly or hold it down as it repeats, the cursor accelerates. It first moves one pixel at a time, then two, then three, until it's moving at the top speed of 12 pixels per keypress. If you stop pressing the cursor key, press another key, or release the key for a moment, the acceleration reverts to one pixel per keypress. If you want fine control, press and release the cursor key slowly, allowing time between each keystroke to prevent acceleration.

Keyboard Shortcuts

Key	Menu
U	Picture/Undo
^N	Picture/New
O	Picture/Open
S	Picture/Save
^Q	Picture/Quit
D	Tools/Draw
L	Tools/Line
R	Tools/Rectangle
C	Tools/Circle
A	Tools/Airbrush
P	Tools/Paint
B	Preferences/Bkgd color
K	Preferences/Keyboard
J	Preferences/Calibrate

Unfortunately, the program is not fast enough to keep up with the full repeat rate of the cursor keys, so even after you release the key, the program is acting on up to 15 pending keystrokes. It's best to control the number of keystrokes yourself by just pressing the same key rapidly rather than holding it down to repeat. If you do hold down the key, release it before the cursor reaches its destination. It will keep going for a short distance, then stop. With practice, you can time things right so that the cursor ends up exactly where you anticipated.

Using The Menus

To access a menu, just move the arrow cursor so that it points at one of the menu titles: *Picture, Tools,* or *Preferences*. The tip of the arrow is the active point, so make sure it is within the menu bar and touching the desired menu title. Now press and release the button. (The keyboard equivalent for the button is the INS key, conveniently located beneath your thumb when you use the cursor keypad.) The menu title reverses color, and the menu drops down.

Note that this differs from the way menus are selected on other machines. On the Atari ST, menus drop down automatically if you merely point at a menu title. On the Macintosh and Amiga, you point at the menu and click to pull it down. You have to continue to hold down the button to keep the menu displayed, and move the pointer within the menu to select an item. To actually make the choice on a Mac or Amiga, you release the mouse button.

In contrast, with Screen Machine II you press and release the button to drop down the menu, move the pointer to the item you want, then press the button again to select the item and remove the menu. This technique is most appropriate when you're using pointing devices such as a joystick or cursor keys, because it's difficult to hold down a button while moving the pointer.

For example, if you point to *Picture*, then click the button, it drops down a list containing the choices *Undo, New, Open, Save, View,* and *Quit* (see figure 3). To the right of each selection are the keyboard equivalents: U, ^N, O, S, V, and ^Q. Instead of pulling down a menu and selecting a choice, you can just press the appropriate keyboard shortcut. The ^ character indicates that you should press the CTRL key along with the following character: ^N means that you should press N while holding down CTRL.

Using the more cumbersome CTRL-N and CTRL-Q sequences, instead of merely N or Q, helps prevent you from casually erasing the screen or exiting the program. Since these are destructive options, the CTRL key is used to guard against accidental keypresses. Use the accompanying table as a quick reference to keyboard shortcuts.

Selecting A Menu Item

To select a menu item, point the cursor at the desired item. As you slide the cursor up or down within a menu, the item you point to is highlighted in reverse video. You then press and release the button to select the highlighted item. To cancel a menu selection, either move the pointer outside of the menu, or move it to point at the menu title so that no other items are selected when you press the button. If you move the cursor to the left or right of the menu border, the menu is automatically canceled. You'll hear an "uh-oh" sound effect to confirm that you've canceled the menu.

When you select an item, on the other hand, it flashes twice, emitting little tweeting sounds to let you know that you've chosen a valid option. (By the way, if you don't want any sound effects, change line 320 to read SNDFX=0.) After you select a menu item, some action is usually performed. If you select *New* from the *Picture* menu, for instance, the screen clears. Use *Quit* to exit the program. Following is a quick tour of the menus—we'll discuss the meaning of each item later on.

Some menu items select a setting for the program. The *Tools* menu contains the choices *Draw, Line, Rectangle, Circle, Airbrush,* and *Paint*. This menu is used to select the current drawing tool. Only one tool is active at any time. When you click the button while pointing at the drawing surface, rather than at the menu bar or within the palette, the current tool is activated, and you start the drawing action. In Draw mode, you can draw connected lines as you move the pointer about. In Line mode, though, you stretch a "rubber-band" line across the screen, emanating from the point you first clicked on. When you press the button a second time, the rubber-band line disappears, and the desired line is stamped down. In Paint mode, each click initiates a flood-fill, used to color enclosed figures.

The *Tools* menu indicates the current tool by placing a check mark next to it. A check mark can show which of several items is currently selected. If you select another drawing tool, the check mark moves to the new tool. In the *Tools* menu, a selection mutually excludes all other selections.

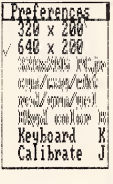
On the other hand, a check mark can also be used to show the status of several on/off settings. The *Preferences* menu lets you select *320 × 200, 640 × 200,* and *320 × 200 PCjr* drawing modes; two color palettes for the 320 × 200 graphics mode: *cyn/mag/wht* and

red/grn/yel; as well as *Bkgd color* (background color), *Keyboard* mode, and *Calibrate*. More than one item can be checked in the *Preferences* menu. You obviously can't draw both in both 320 × 200 mode and 640 × 200 mode at the same time, so only one of the three graphics modes is checked. However, while in 320 × 200 mode, you can choose either of the two color palettes, so both 320 × 200 would be checked as well as either *cyn/mag/wht* or *red/grn/yel*.

Ghosted Items

If you select 640 × 200 mode, some menu items under *Preferences* are no longer appropriate. It isn't possible to switch color palettes or change the background color while in 640 × 200 mode, so the inappropriate menu selections need to be disabled. Also, the 320 × 200 PCjr mode, which permits 16 colors, only works with the PCjr, so this selection should be made inaccessible when running on a PC.

Figure 1: Ghosted Items



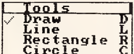
In 640 × 200 mode, the menu items *cyn/mag/wht, red/grn/yel,* and *Bkgd color* are disabled, and the text of the menu items is distorted to show that you can't access them (see Figure 1). This distortion dims and garbles the text—such a menu item is *ghosted out*, as if the text was a "ghost" of the original text. The 320 × 200 *PCjr* option is ghosted out when running on the PC. A ghosted menu item can't be selected; you can't even highlight it by pointing to it.

A ghosted item immediately tells you that the menu item is inappropriate for the current environment. If you wonder which commands work in which modes, ghosting makes it obvious. Along with ghosting and the check mark, a menu can be documentation, online help, and a status report, as well as the device used to change these settings or activate commands.

Drawing Tools

The pointer arrow is the pass key to all the functions of Screen Machine II. You use it to select a menu item, sketch on the drawing surface, or choose a new drawing color. You already know how to use the menus. To change drawing colors, just point within the box containing the desired color and click the button. A border encloses the current drawing color so you can tell at a glance which color is being used. In 640 × 200 mode, you can only switch between black and white, of course. On the PCjr, you can select the 16-color drawing mode (see below).

Figure 2: The *Tools* Menu



Your default drawing tool is *Draw*, as you can confirm by looking within the *Tools* menu. With all the tools, you start the drawing action by clicking the button while pointing at the drawing area. The drawing area is bordered by the menu bar, the color palette, and is enclosed within a rectangle. You can't draw anywhere outside of this border.

In Draw mode, the first button click initiates the mode. The cursor disappears (to speed up drawing), and a point is plotted at the cursor position. You can now move around on the screen, leaving a trail behind, as you sketch freehand with the joystick, tablet, or cursor keys. The cursor keys are especially useful for touch-up work or small, complex figures. Again, you can use the *Keyboard* option from *Preferences* if you need to disable the joystick. If you press the button (or merely lift the stylus from the tablet surface), you exit drawing mode, and you can once again freely move the cursor without drawing on your screen canvas.

In *Line, Rectangle,* and *Circle* modes, the first click sets the first coordinate for the figure. You then move the pointer about to change the size or position of the figure, then press the button again to finalize the figure. While previewing the figure, the line, circle, or rectangle is repeatedly drawn and erased to allow movement. As you move the figure across the drawing surface, it may erase parts of the picture as it passes over the screen. Don't be concerned—this is just a side effect of the animation process. When you press the button to choose the desired figure, the previous screen is redrawn, restoring any erased parts, and then the desired figure is overlaid on top of the picture. If you make a mistake, you can use the *Undo* option from the *Picture* menu (or simply press U), to restore the previous screen.

In *Line* mode, the first click sets one endpoint of the line. You move the other endpoint around with your pointing device. You see how the line will look as you move it around the screen. In *Rectangle* mode, the first click sets one corner of the rectangle. As you move the pointer, you are dragging around the diagonally opposite corner of the rectangle. In *Circle* mode, the first click sets the center of the circle. You move the pointer around to enlarge or contract the radius of the circle. The second click stamps down the figure. (Remember, you can always Undo the most recent drawing action.)

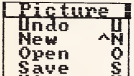
The *Airbrush* tool is handy for shading or blending colors. It randomly sprays out pixels within an 8 × 8 zone centered around the pointing arrow. The longer you stay in one place, the more paint is sprayed down. If you move around while spraying, you get a series of random dots. This approximates the behavior of a real airbrush. Again, the first click starts airbrush mode, and a second click exits airbrush mode, restoring floating cursor movement. As with *Draw*

mode, lifting the stylus from the tablet surface implies you want to stop the airbrush and go back to moving the cursor.

Use the *Paint* tool carefully. It's used to fill in an enclosed area of the screen. For example, you could draw a rectangle first, then fill it in. The paint floods out of the cursor position, and doesn't stop until it touches areas of the same color. You can only fill an area bounded by the same color as the current drawing color. If you attempt to fill with a different color, the paint overflows the container, possibly filling the whole screen. However, if you remember to press U before you start another drawing action, no harm is done.

You can use the keyboard shortcuts D for *Draw*, L for *Line*, R for *Rectangle*, C for *Circle*, A for *Airbrush*, and P for *Paint*.

Figure 3: The *Picture* Menu



The *Picture* Menu

The *Picture* menu affects the overall screen canvas. Use *Undo* to restore the previous screen. The screen is saved in a buffer before any drawing command changes the screen. *Undo* copies this buffer back onto the screen, restoring the previous screen and erasing the most recent change. Of course, *Undo* can only undo the most recent action, and you can't go back to the way the screen was before you performed the Undo—you can't undo an *Undo*.

The *New* option simply erases the screen. Be careful—it doesn't ask "Are you sure?" first.

Use *Quit* to exit the program. You could, of course, simply turn off the machine, but *Quit* is somewhat more elegant. Once you're back in BASIC, you can type SYSTEM to exit to DOS.

The *Save* command stores your picture on disk. *Open* restores a previously saved screen. After you select *Open* or *Save*, a box pops up in the center of the screen, prompting you to enter a filename. You can enter any legal PC-DOS filename, including a path prefix, such as A: or B:. This is the name that your picture is stored under. After you *Save* a picture, you can use *Open* to read this picture back onto the screen.

If you don't use an extender, as in FLOWERS.ART, an extender is added for you. The extender is made of the characters PI (for picture) and the number of the graphics mode used: 1 for SCREEN 1, 320 × 200; 2 for SCREEN 2, 640 × 200, and 5 for the PCjr 16-color 320 × 200 mode, SCREEN 5.

So a picture saved while in 640 × 200 mode would have the characters .PI2 appended to the filename. This extender is added for both Open and Save. If you attempt to Open the picture FLOWERS while in 320 × 200 mode, it actually searches for FLOWERS.PI1. However, if you're in 640 × 200 mode, it searches for FLOWERS.PI2. This prevents you from loading a picture saved in one graphics mode onto the screen of another graphics mode. If you want to defeat this, either always use an extension, as in FLOWERS.ART, or append the appropriate .PI extension. If you're loading a picture saved as FLOWERS.PI1 onto the 640 × 200 screen, you need to enter the filename FLOWERS.PI1 to prevent it from searching for FLOWERS.PI2.

If a disk error occurs, another box pops up showing you the DOS error code, and it prompts you to press either R for Retry or C for Cancel. If the error is something you can immediately correct, such as inserting a disk, you can press R to retry the disk operation. Otherwise press C to cancel the operation, then figure out what you did wrong before again selecting Open or Save.

Next month, we'll show you how to use the BLOAD command to load one of these pictures from within your own programs. If you can't wait, examine the Open and Save routines at lines 2100 and 2170.

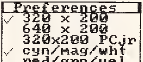
The keyboard shortcuts for the *Picture* menu are ^N for *New*, O for *Open*, S for *Save*, and ^Q for *Quit* (remember that ^ means to press CTRL as you press the indicated key).

Choosing Your Preferences

The *Preferences* menu lets you select various special options. The first three entries: *320 × 200*, *640 × 200*, and *320 × 200 PCjr*, let you pick which graphics mode to use. (You should choose your mode before you begin drawing; the drawing area is erased when you change modes, so any drawing will be lost.) The 320 × 200 mode gives you four colors to work with. In this mode, you can choose either of two color palettes: *cyn/mag/wht* (cyan/magenta/white), or *red/grn/yel* (red/green/yellow). The latter options are ghosted in all other modes.

The 640 × 200 mode gives you more horizontal density for fine detail, but you can only choose between black and white. On the PCjr only, you can select the *320 × 200 PCjr* mode and get 16 colors for vivid, realistic (or surrealistic) paintings.

Figure 4: The *Preferences* Menu



The *Bkgd color* option switches to a different background color. Each time you select it, the background color changes to the next in a series, 16 colors in all.

Some of the *Preferences* items have keyboard shortcuts: B for *Bkgd color*, K for *Keyboard*, and J for *Calibrate*.

BASIC Shortcomings

This program was an experiment of sorts, an attempt to discover if techniques such as pull-down menus can be achieved in BASIC. I knew from the beginning that BASIC's relatively slow speed (as compared to machine language or compiled languages) would be the limiting factor. In particular, using a 30,000-byte array for an Undo buffer causes a short delay the first time any routine is activated. The first time you try to move the cursor,

select a drawing tool, change colors, select a menu, etc., there is short delay as the huge array is shifted downward in memory to make room for new variables as they are encountered. This problem could be eliminated by referencing every variable in the program before the array is dimensioned, but this is really more trouble than it's worth.

The innermost, core routine in this program is the subroutine at line 20000, used to check for the "mouse" position. It adapts to the keyboard, joystick, and touch tablet, checks for keyboard shortcuts, scales the values to the current screen resolution, and keeps these coordinates in bounds. All this checking in such a commonly called routine is bound to slow things down. If you are using only one pointing device or one graphics mode, you may want to consider streamlining this routine to speed things up.

Screen Machine II

For instructions on entering this listing, please refer to "COMPUTE!'s Guide to Typing In Programs" in this issue of COMPUTE!

```
HA 120 OEFINT A-Z
PA 140 PCJR=0:DN ERRDR GOTO 150:
       SOUND OFF:CLEAR ,,,32768!
       :DEFINT A-Z:PCJR=-1
JF 150 IF NOT PCJR THEN RESUME 1
       60
OJ 160 ON ERROR GOTO 0
CN 220 OIM ARRDW%(32),ZZTEMP%(64
       8)
KF 240 XMAX=250:YMAX=230:XOFF=7:
       YOFF=7
HG 250 HIGHLIGHT=2
ND 260 TRUE=-1:CURSOR=TRUE
MP 270 KEY OFF:SCREEN 0,0,0:WIOT
       H 40:COLOR ,1,1:CLS:LOCAT
       E 4,11,0:COLOR 12:PRINT "
       SCREEN MACHINE II"
MP 280 LOCATE 7,12:COLOR 10:PRIN
       T"Charles Brannon"
KD 290 COLOR 14:LOCATE 13,10:PRI
       NT"One moment, please..."
AC 300 GOSUB 9000
OB 310 SMOOE=1:COLR=1:GOSUB 3000
BL 320 SNOFX=TRUE
AG 330 ACC=1:OACC=1
KG 340 FROZEN=0
BH 380 COLR=1:TOOL=1
EO 390 STRIG ON
PC 400 MX=XRES/2:MY=YRES/2:NX=MX
       :NY=MY:GOSUB 18000
LI 410 OIM UNOO%(15000)
JI 450 WHILE TRUE
BN 460  GOSUB 18000:MB=0:MNIO=0
LF 470  WHILE MNIO=0 ANO MB=0
KF 480   GOSUB 14000
BH 490  WENO
FN 500  IF MB<>0 THEN GOSUB 1000
FN 510  IF MNIO THEN GOSUB 2000
EM 520 WENO
LM 1000 WHILE MB:GDSUB 20000:WEN
        D
KB 1010 GOSUB 19000
AC 1020 IF MY>=CY THEN COLR=INT(
        MX/XR#):GOSUB 6000:RETUR
        N
IC 1030 GET (1,8)-(XRES-2,CY-1),
        UNDD%
OJ 1035 SCM$=CM$:CM$=""
LG 1040 DN TDDL GOSUB 1070,1170,
        1300,1430,1560,1630
OC 1045 CM$=SCM$
JK 1050 RETURN
EA 1070 IF PENUP AND NDT KEYMDDE
         THEN RETURN
KB 1080 CURSDR=0
GA 1090 WHILE MB=0 AND (NDT PENU
        P OR KEYMDDE)
EG 1100 SX=MX:SY=MY:GOSUB 20000:
        MY=-MY*(MY>7 AND MY<CY)-
        B*(MY<B)-(CY-1)*(MY>=CY)
ML 1110 LINE (SX,SY)-(MX,MY),COL
        R
FK 1120 WEND
LH 1130 WHILE MB:GOSUB 20000:WEN
        D
HC 1140 CURSOR=TRUE
JM 1150 RETURN
DF 1170 SX=MX:SY=MY:CURSOR=0
MP 1180 WHILE MB=0
OO 1190  LINE (SX,SY)-(MX,MY),0
OG 1200  GOSUB 20000:MY=-MY*(MY>
        7 AND MY<CY)-B*(MY<B)-(C
        Y-1)*(MY>=CY)
BK 1210  LINE (SX,SY)-(MX,MY),CO
        LR
EA 1220  EX=MX:EY=MY
FP 1230 WEND
MM 1240 WHILE MB:GOSUB 20000:WEN
        O
NA 1250 PUT (1,8),UNOO%,PSET
HE 1260 LINE (SX,SY)-(EX,EY),COL
        R
IN 1270 CURSDR=TRUE
JH 1280 RETURN
CE 1300 SX=MX:SY=MY:CURSOR=0
LO 1310 WHILE MB=0
NK 1320  LINE (SX,SY)-(MX,MY),0,
        B
PB 1330  GOSUB 20000:MY=-MY*(MY>
        7 AND MY<CY)-B*(MY<B)-(C
        Y-1)*(MY>=CY)
FH 1340  LINE (SX,SY)-(MX,MY),CO
        LR,B
FL 1350  EX=MX:EY=MY
GK 1360 WEND
MH 1370 WHILE MB:GOSUB 20000:WEN
        O
OL 1380 PUT (1,8),UNDD%,PSET
FD 1390 LINE (SX,SY)-(EX,EY),COL
        R,B
HM 1400 CURSOR=TRUE
IG 1410 RETURN
DP 1430 SX=MX:SY=MY:CURSOR=0
MJ 1440 WHILE MB=0
PM 1450  CIRCLE (SX,SY),SQR(ABS(
        SX-MX)^2+ABS(SY-MY)^2),0
QM 1460  GOSUB 20000:MY=-MY*(MY>
        7 AND MY<CY)-B*(MY<B)-(C
        Y-1)*(MY>=CY)
NP 1470  CIRCLE (SX,SY),SQR(ABS(
        SX-MX)^2+ABS(SY-MY)^2),C
        OLR
FG 1480  EX=MX:EY=MY
GF 1490 WEND
LG 1500 WHILE MB:GOSUB 20000:WEN
        D
KE 1510 GOSUB 3000:PUT (1,8),UNO
        O%,PSET
JA 1520 CIRCLE (SX,SY),SQR(ABS(S
        X-EX)^2+ABS(SY-EY)^2),CO
        LR
MN 1530 CURSOR=TRUE:GDSUB 12000:
        GDSUB 6000
JB 1540 RETURN
GB 1560 WHILE MB=0 AND (NDT PENU
        P OR KEYMDDE)
JE 1570  GDSUB 20000:IF MY<12 DR
         MY>CY-5 THEN 1590
BG 1580  GDSUB 19000:PSET (MX+4-
        B*RND,MY+4-B*RND),COLR
GH 1590 WEND
OI 1600 WHILE MB:GDSUB 20000:WEN
        D
JK 1610 RETURN
NA 1630 DN ERROR GDTD 1660:PAINT
         (MX,MY),CDLR:LINE (0,0)
        -(XRES-1,YRES-1),,B:GDSU
        B 6000:GDSUB 12000
GC 1640 ON ERRDR GOTD 0:WHILE MB
        :GDSUB 20000:WEND
JG 1650 RETURN
KA 1660 RESUME NEXT
ED 2000 ON MNIO GOSUB 2030,2320,
        2380
IP 2010 RETURN
GL 2030 ON MNIT GOSUB 2060,2080,
        2100,2170,2240,2300
JI 2040 RETURN
AF 2060 GOSUB 19000:PUT (1,8),UN
        OO%,PSET:RETURN
HE 2080 GOSUB 3000:RETURN
IJ 2100 TYP$="DPEN":GOSUB 4000
OJ 2110 IF FILENAME$="" THEN 213
        0
NC 2120 ON ERRDR GOTO 5500:OEF S
        EG=SEGAOR:BLOAD FILENAME
        $,0
JD 2130 ON ERROR GOTO 0:CLOSE#1
BD 2140 LINE (0,0)-(XRES-1,YRES-
        1),,B:GOSUB 12000:GOSUB
        6000
JN 2150 RETURN
GC 2170 TYP$="SAVE":GOSUB 4000
NA 2180 IF FILENAME$="" THEN 221
        0
FN 2190 GET (1,8)-(XRES-2,CY-1),
        UNOO%:CLS:PUT (1,8),UNDO
        %,PSET
OL 2200 ON ERROR GOTO 5500:DEF S
        EG=SEGADR:BSAVE FILENAME
        $,0,SCRLEN'
KP 2210 ON ERROR GOTO 0:CLOSE#1:
        GOSUB 3000:PUT (1,8),UNO
        O%,PSET
IG 2220 RETURN
OF 2240 GOSUB 19000:CURSOR=0
EO 2250 GET (1,8)-(XRES-2,CY-1),
        UNOO%:CLS:PUT (1,8),UNDO
        %,PSET
HB 2260 WHILE MB=0:GDSUB 20000:W
        ENO
OG 2270 WHILE MB:GOSUB 20000:WEN
        O
MF 2280 GOSUB 3000:PUT (1,8),UNO
        O%,PSET:CURSOR=-1:RETURN
CJ 2300 SCREEN 0,0,0,0:END
LM 2320 MFLAGS(MNIO,TOOL)=1
OH 2330 MFLAGS(MNID,MNIT)=2:TODL
        =MNIT
JO 2340 RETURN
BN 2350 STOP
MF 2380 IF MNIT<4 THEN SMOOE=MNI
        T-2*(MNIT=3):GOSUB 3000
PK 2390 IF MNIT=4 THEN COLOR ,1:
        MFLAGS(MNID,4)=2:MFLAGS(
        MNIO,5)=1
BM 2400 IF MNIT=5 THEN COLOR ,2:
        MFLAGS(MNIO,4)=1:MFLAGS(
        MNIO,5)=2
IF 2410 IF MNIT=6 THEN BG=(BG+1)
         AND 15:IF SMODE=1 THEN
        COLOR BG ELSE COLOR ,BG
GF 2420 IF MNIT=7 THEN FROZEN=NO
        T FROZEN:MFLAGS(MNIO,MNI
        T)=1-FROZEN
```

```
DE 243Ø IF MNIT<>B THEN RETURN
GA 244Ø GDSUB 19ØØØ:LDCATE 1,1:M
        SG$=LEFT$("Move stick to
         upper left, press butto
        n."+SPACE$(BØ),SWIDTH):G
        DSUB 13ØØØ
IJ 245Ø WHILE STRIG(1)=Ø:XOFF=ST
        ICK(Ø):YDFF=STICK(1):WEN
        D
MM 246Ø WHILE STRIG(1)<>Ø:WEND
FO 247Ø LOCATE 1,1:MSG$=LEFT$("M
        ove stick to lower right
        , press button."+SPACE$(
        BØ),SWIDTH):GDSUB 13ØØØ
HK 248Ø WHILE STRIG(1)=Ø:XMAX=ST
        ICK(Ø):YMAX=STICK(1):WEN
        D
MG 249Ø WHILE STRIG(1)<>Ø:WEND
DH 25ØØ XRATIO#=XRES/XMAX:YRATIO
        #=YRES/YMAX
IF 251Ø GOSUB 12ØØØ:RETURN
KA 3ØØØ GOSUB 19ØØØ
KO 3Ø1Ø IF SMODE=PMODE THEN 3Ø3Ø
KD 3Ø2Ø DN SMODE GDSUB 311Ø,315Ø
        ,3Ø3Ø,3Ø3Ø,319Ø
CP 3Ø3Ø PMODE=SMODE
AG 3Ø4Ø SWIDTH=INT(XRES/B):XRATI
        O#=XRES/XMAX:YRATIO#=YRE
        S/YMAX
CG 3Ø5Ø CLS:PSET (1Ø,1Ø):ORAW "b
        m1Ø,1Ød3e313f5":GET (1Ø,
        1Ø)-(17,17),ARROW%
HI 3Ø6Ø XARROW=B:YARRDW=B
DL 3Ø7Ø CLS:LINE (Ø,Ø)-(XRES-1,Y
        RES-1),,B
PC 3ØBØ GOSUB 6ØØØ:GOSUB 12ØØØ
KI 3Ø9Ø RETURN
DD 311Ø SCREEN 1:COLOR Ø,1:COLR=
        1:XRES=32Ø:YRES=2ØØ:BG=Ø
        :MAXCOLDR=4
HA 312Ø GOSUB 323Ø:MFLAGS(3,1)=2
        :SEGAOR=&HBBØØ:SCRLEN'=1
        63B4
PJ 313Ø MFLAGS(3,4)=2:MFLAGS(3,5
        )=1:MFLAGS(3,6)=1
JL 314Ø RETURN
QN 315Ø SCREEN 2:XRES=64Ø:YRES=2
        ØØ:MAXCDLDR=2:CDLR=1
JG 316Ø GOSUB 323Ø:MFLAGS(3,2)=2
        :SEGADR=&HB8ØØ:SCRLEN'=1
        63B4
ID 317Ø MFLAGS(3,4)=Ø:MFLAGS(3,5
        )=Ø:MFLAGS(3,6)=Ø
JH 31BØ RETURN
JK 319Ø SCREEN 5:XRES=32Ø:YRES=2
        ØØ:MAXCOLOR=16:CDLR=1
FE 32ØØ GOSUB 323Ø:MFLAGS(3,3)=2
        :SEGADR=&H1BØØ:SCRLEN'=3
        276B'
KB 321Ø MFLAGS(3,4)=Ø:MFLAGS(3,5
        )=Ø:MFLAGS(3,6)=1
IH 322Ø RETURN
JC 323Ø MFLAGS(3,1)=1:MFLAGS(3,2
        )=1:MFLAGS(3,3)=-PCJR:RE
        TURN
DL 4ØØØ GOSUB 19ØØØ:GET (1,B)-(X
        RES-2,CY-1),UNOD%
JO 4Ø1Ø MSG1$="Please enter name
        ":MSG2$="of picture to "
        +TYP$
FF 4Ø2Ø TW=SWIDTH/2-1Ø:LINE (TW*
        B-1Ø,5Ø)-(TW*B+16Ø,1ØØ),
        Ø,BF:LINE (TW*B-1Ø,5Ø)-(
        TW*B+16Ø,1ØØ),,B:LINE (T
        W*B-B,52)-(TW*B+15B,9B),
        ,B
HD 4Ø3Ø LOCATE B,SWIDTH/2-LEN(MS
        G1$)/2:PRINT MSG1$:LOCAT
        E 9,SWIOTH/2-LEN(MSG2$)/
        2:PRINT MSG2$
PG 4Ø4Ø LINE (TW*B-5,7B)-(TW*B+1
        55,B9),,B:LOCATE 11,TW+1
        :MAXLEN=1B:GOSUB 5ØØØ
IP 4Ø5Ø FILENAME$=EDT$:IF FILENA
        ME$>"" THEN IF MID$(EDT$
        ,LEN(EDT$)+3*(LEN(EDT$)>
        3),1)<>"." THEN FILENAME
        $=FILENAME$+".PI"+CHR$(4
        B+SMDDE)
NC 4Ø6Ø PUT (1,B),UNDD%,PSET
JD 4Ø7Ø RETURN
QP 5ØØØ EDT$="":IX=PDS(Ø):IY=CSR
        LIN:XI=IX:KBD=-1:IF MAXL
        EN=Ø THEN MAXLEN=79-IX
GB 5Ø1Ø WHILE KBO<>13
GE 5Ø2Ø  XI=LEN(EOT$)+IX:LOCATE
        IY,XI:PRINT "_";:KBO$=IN
        PUT$(1)
PF 5Ø3Ø  KBO=ASC(KBO$):LOCATE IY
        ,XI:PRINT" ";
DC 5Ø4Ø  IF KBO=B AND LEN(EOT$)>
        Ø THEN EOT$=LEFT$(EOT$,L
        EN(EDT$)-1)
KH 5Ø5Ø  IF LEN(EOT$)<MAXLEN AND
         (KBO AND 127)>=32 THEN
        EDT$=EOT$+KBO$:LDCATE IY
        ,XI:PRINT KBD$;
GI 5Ø6Ø WEND
JE 5Ø7Ø RETURN
JL 55ØØ CLOSE #1
EJ 551Ø GDSUB 19ØØØ:GET (1,B)-(X
        RES-2,CY-1),UNDD%
GA 552Ø TW=SWIOTH/2-1Ø:LINE (TW*
        B-1Ø,5Ø)-(TW*B+16Ø,1ØØ),
        Ø,BF:LINE (TW*B-1Ø,5Ø)-(
        TW*B+16Ø,1ØØ),,B:LINE (T
        W*B-B,52)-(TW*B+15B,9B),
        ,B
PK 553Ø IF ERR>=52 THEN MSG1$="D
        OS ERROR #"+STR$(ERR):EL
        SE MSG1$="ERRDR #"+STR$(
        ERR)+" in Iine"+STR$(ERL
        )
JF 554Ø MSG2$="(R)etry or (C)anc
        el"
PD 555Ø LOCATE B,SWIDTH/2-LEN(MS
        G1$)/2:PRINT MSG1$:LDCAT
        E 1Ø,SWIDTH/2-LEN(MSG2$)
        /2:PRINT MSG2$
BF 556Ø KBO$=INPUT$(1):IF KBD$<>
        "r" AND KBD$<>"R" AND KB
        O$<>"c" AND KBO$<>"C" TH
        EN 556Ø
OA 557Ø PUT (1,B),UNOO%,PSET
HL 55BØ IF KBD$="r" OR KBO$="R"
        THEN RESUME ELSE RESUME
        NEXT
FH 6ØØØ XR#=XRES/MAXCOLOR:CH=11:
        CY=YRES-CH-1
PH 6Ø1Ø LINE (Ø,CY)-(XRES-1,YRES
        -1),Ø,BF
EJ 6Ø2Ø FOR I=Ø TO MAXCOLOR-1
DA 6Ø3Ø LINE (I*XR#+2,CY+3)-(I*X
        R#+XR#-3,CY+CH-3),I,BF
DN 6Ø4Ø NEXT
BH 6Ø5Ø LINE (Ø,CY)-(XRES-1,YRES
        -1),,B
JD 6Ø6Ø LINE (COLR*XR#,CY+2)-(CO
        LR*XR#+XR#-1,CY+CH-2),,B
JF 6Ø7Ø RETURN
BK 9ØØØ RESTDRE 9Ø9Ø
GO 9Ø1Ø WHILE MNSTR$<>"x"
MJ 9Ø2Ø  REAO MNIO,MNIT,MFLAG,MN
        STR$
ME 9Ø3Ø  IF MNSTR$<>"x" THEN GDS
        UB 11ØØØ
FG 9Ø4Ø WEND
CM 9Ø5Ø MFLAGS(3,3)=-PCJR
BM 9Ø6Ø CM$="U11"+CHR$(14)+"12O1
        3S14V15"+CHR$(17)+"16O21
        L22R23C24A25P26B36K37J3B
        "
KI 9Ø7Ø RETURN
BD 9Ø9Ø OATA 1,Ø,1,"Picture "
HE 91ØØ OATA 1,1,1,"Undo   U"
IN 911Ø DATA 1,2,1,"New   ^N"
LH 912Ø DATA 1,3,1,"Dpen   D"
PN 913Ø DATA 1,4,1,"Save   S"
EM 914Ø DATA 1,5,1,"View   V"
EK 915Ø DATA 1,6,1,"Quit  ^Q"
MD 917Ø DATA 2,Ø,1," Tools
        "
HJ 91BØ DATA 2,1,2," Draw      D
        "
ML 919Ø DATA 2,2,1," Line      L
        "
PM 92ØØ DATA 2,3,1," Rectangle R
        "
ID 921Ø OATA 2,4,1," Circle    C
        "
JC 922Ø DATA 2,5,1," Airbrush  A
        "
PI 923Ø OATA 2,6,1," Paint     P
        "
BP 925Ø DATA 3,Ø,1,"Preferences
        "
HG 926Ø DATA 3,1,2," 32Ø x 2ØØ
        "
OI 927Ø DATA 3,2,1," 64Ø x 2ØØ
        "
KD 92BØ DATA 3,3,Ø," 32Øx2ØØ PCj
        r"
DB 929Ø DATA 3,4,2," cyn/mag/wht
        "
JI 93ØØ DATA 3,5,1," red/grn/yel
        "
JM 931Ø DATA 3,6,1," Bkgd color
        B"
ED 932Ø DATA 3,7,1," Keyboard
        K"
IB 933Ø DATA 3,B,1," Calibrate
        J"
OD 934Ø DATA ,,,x
LJ 11ØØØ MAXMENUS=B:MAXITEMS=B
GF 11Ø1Ø IF NOT MENUINIT THEN OI
        M MTITLE$(MAXMENUS,MAXI
        TEMS),MFLAGS(MAXMENUS,M
        AXITEMS),MITEMS(MAXMENU
        S),MSAVE%(BØØ*MAXITEMS+
        B),MX(MAXMENUS):TDPID=Ø
        :MENUINIT=-1
JE 11Ø2Ø IF MNIO<1 OR MNIO>MAXME
        NUS OR MNIT<Ø OR MNIT>M
        AXITEMS THEN PRINT "ILL
        EGAL MENU PARAMETERS":S
        TOP
IH 11Ø3Ø MTITLE$(MNID,MNIT)=MNST
        R$:MFLAGS(MNIO,MNIT)=MF
        LAG
DL 11Ø4Ø IF MNIT>MITEMS(MNID) TH
        EN MITEMS(MNID)=MNIT
BA 11Ø5Ø IF MNIO>TOPID THEN TOPI
        O=MNIO
IF 11Ø6Ø RETURN
QJ 12ØØØ IF SWIOTH=Ø THEN IF XSI
        ZE THEN SWIOTH=INT(XSIZ
        E/B+.5) ELSE SWIDTH=BØ
DM 12Ø1Ø MSG$=" ":MX(Ø)=B:SVX=PO
        S(Ø):SVY=CSRLIN
GL 12Ø2Ø FOR MI=1 TO TOPIO:MX(MI
        )=MX(MI-1)+B+LEN(MTITLE
        $(MI,Ø))*B:MSG$=MSG$+"
        "+MTITLE$(MI,Ø):NEXT:MS
        G$=MSG$+SPACE$(SWIOTH-L
        EN(MSG$))
PF 12Ø3Ø LOCATE 1,1:GOSUB 13ØØØ
HC 12Ø4Ø LOCATE SVY,SVX:RETURN
BI 13ØØØ X1=PDS(Ø)*B-B:Y1=CSRLIN
        *B-B:PRINT MSG$;:X2=X1+
        LEN(MSG$)*B-1:IF X2>=SW
        IOTH*B THEN X2=SWIDTH*B
        -1
LL 13Ø1Ø GET (X1,Y1)-(X2,Y1+7),Z
        ZTEMP%:PUT (X1,Y1),ZZTE
        MP%,PRESET:RETURN
MD 14ØØØ XSAVE=POS(Ø):YSAVE=CSRL
        IN
DC 14Ø1Ø MNIT=Ø:MNID=Ø:GOSUB 2ØØ
        ØØ
```

```
LM 14020 IF MY>7 OR MB=Ø THEN RETURN
FM 14030 WHILE MB:GDSUB 2ØØØØ:WEND
EH 14040 MI=1:WHILE MI<=TOPID AND NDT (MX>=MX(MI-1) AND MX<=MX(MI)):MI=MI+1:WEND
DG 14050 IF MI>TOPID THEN RETURN
CF 14060 MNID=MI
HJ 14070 IF SNDFX THEN SDUND 1ØØØØ,.5
QI 14080 GDSUB 16ØØØ:GDSUB 2ØØØØ
FE 14090 SAVDACC=DACC:SAV$=CM$:CM$="":IF KEYMDDE THEN MY=2:NY=MY:DACC=-8
DN 14100 WHILE MX>=MX(MNIO-1) AND MX<=MX(MNID) AND MB=Ø
KI 14110  GDSUB 2ØØØØ
HD 14120  MI=INT(MY/B):IF MI>MITEMS(MNID) THEN GDTD 1415Ø
AF 14130  IF MI=MNIT OR MFLAGS(MNID,MI)=Ø THEN 141BØ
CN 14140  GDSUB 19ØØØ
OL 14150  IF MNIT>Ø THEN LDCATE MNIT+1,INT(MX(MNID-1)/B+2):PRINT MTITLE$(MNID,MNIT)
LG 14160  IF MI>Ø AND MI<=MITEMS(MNID) THEN MNIT=MI:LDCATE MNIT+1,INT(MX(MNIO-1)/B)+2:MSG$=MTITLE$(MNID,MNIT):GDSUB 13ØØØ:IF SNDFX THEN SDUND 2ØØØØ,.1
OD 14170 IF MI>MITEMS(MNID) THEN MNIT=Ø
LL 141B0 WEND
BE 14190 IF MX<MX(MNID-1) OR MX>MX(MNID) THEN MNIT=Ø
HE 14200 IF MNIT THEN GDSUB 15ØØØ
GO 14210 GOSUB 17ØØØ
FO 14220 WHILE MB:GDSUB 2ØØØØ:WEND
NG 14240 IF MNIT=Ø THEN MNID=Ø:IF SNDFX THEN SDUND 15Ø,2:SDUND 5Ø,1
CK 14250 GOSUB 1BØØØ:OACC=SAVOACC:CM$=SAV$:LDCATE YSAVE,XSAVE
JB 14260 RETURN
JM 15000 IF MNIT=Ø OR HIGHLIGHT=Ø THEN RETURN
PC 15010 MSG$=MTITLE$(MNIO,MNIT):FDR MI=1 TO HIGHLIGHT:LDCATE MNIT+1,XP:GOSUB 13ØØØ
ON 15020 IF SNDFX THEN SDUND 1ØØØØ+MI*5ØØ,.1
LA 15030 LOCATE MNIT+1,XP:PRINT MSG$
BN 15040 NEXT:RETURN
KE 16000 WX1=MX(MNIO-1):WX2=MX(MNIO):WY1=8:WY2=B+B*MITEMS(MNIO):XP=INT(WX1/B)+2
II 16010 GOSUB 19ØØØ
CL 16020 LOCATE 1,XP-1:PRINT " "+MTITLE$(MNIO,Ø)
EE 16030 GET (WX1-2,WY1)-(WX2+2,WY2+2),MSAVE%
NN 16040 LINE (WX1-2,WY1-1)-(WX2+2,WY2+2),,B
IL 16050 LINE (WX1-1,WY1)-(WX2+1,WY2+1),Ø,BF
MM 16060 FOR MI=1 TO MITEMS(MNIO)
JO 16070  LOCATE MI+1,XP:PRINT MTITLE$(MNID,MI)
DL 16080  IF MFLAGS(MNID,MI)=2 THEN PSET (WX1,MI*B+5):DRAW "f2e5"
CA 16090  IF MFLAGS(MNID,MI)=Ø THEN GET (WX1,MI*B)-(WX1+LEN(MTITLE$(MNID,MI))*B+7,MI*B+7),ZZTEMP%:PUT (WX1,MI*8),ZZTEMP%,PSET:PUT (WX1+1,MI*B),ZZTEMP%
QL 16100 NEXT MI
IO 16110 RETURN
HG 17000 GDSUB 19ØØØ
QK 17010 PUT (WX1-2,WY1),MSAVE%,PSET
AB 17020 LDCATE 1,XP-1:MSG$=" "+MTITLE$(MNID,Ø):GDSUB 13ØØØ
IF 17030 RETURN
LA 18000 IF CURSOR=Ø DR TDGGLE=1 THEN RETURN
HJ 18010 PUT (MX,MY),ARRDW%:TDGGLE=1:RETURN
JF 19000 IF CURSDR=Ø DR TDGGLE=Ø THEN RETURN
FK 19010 PUT (MX,MY),ARRDW%:TDGGLE=Ø:RETURN
DN 20000 MB=Ø:PENUP=Ø
JL 20010 IF NDT FROZEN THEN SØ=STICK(Ø):S1=STICK(1):M8=STRIG(1):IF SØ<>XDFF DR S1<>YDFF THEN NX=INT((SØ-XOFF)*XRATIO#):NY=INT((S1-YDFF)*YRATID#):KEYMDDE=Ø:ELSE PENUP=-1
JN 20020 MK$=INKEY$:KY=Ø:IF MK$="" THEN IF  TIMER>=TM! THEN ACC=ABS(DACC):TM!=TIMER+.1:GOTD 2ØØ6Ø ELSE 2ØØ6Ø
LM 20025 KY=ASC(MID$(MK$,2)+CHR$(Ø)):M8=M8 DR -(KY=82):KEYMDDE=-1
EE 20030 NX=-(NX+ACC*(KY=75)-ACC*(KY=77))*(KY<>71):NY=-(NY+ACC*(KY=72)-ACC*(KY=BØ))*(KY<>71)
MA 20040 IF KY=PK THEN ACC=ACC+2*(ACC<13)*(DACC>Ø):PK=KY:ELSE ACC=ABS(OACC):PK=KY
DC 20050 KY=ASC(MK$):IF NOT (KY>47 AND KY<5B) THEN WHERE=INSTR(CM$,CHR$(KY+32*(KY>96 AND KY<123))):IF WHERE THEN MNID=VAL(MIO$(CM$,WHERE+1,1)):MNIT=VAL(MIO$(CM$,WHERE+2,1)):IF MFLAGS(MNID,MNIT)=Ø THEN MNIT=Ø:MNID=Ø ELSE GDSUB 21Ø1Ø
HK 20060 IF NX=MX AND NY=MY THEN RETURN
DP 20065 XBDUNO=XRES-XARROW:Y8OUNO=YRES-YARROW
EI 20070 NX=-NX*(NX>Ø AND NX<=XBOUNO)-XBOUND*(NX>XBOUNO)-(NX<1)
FO 200B0 NY=-NY*(NY>Ø AND NY<=Y8DUND)-YBOUND*(NY>YBDUNO)-(NY<1)
PN 20090 GOSUB 19ØØØ:MX=NX:MY=NY:GOSUB 18ØØØ
HP 20100 RETURN
JJ 21010 XP=INT(MX(MNID-1)/B)+2:MSG$=" "+MTITLE$(MNIO,Ø):GOSU8 19ØØØ
GG 21015 LOCATE 1,XP-1:PRINT MSG$
NI 21020 IF SNOFX THEN SOUNO 1ØØØØ,.1
FL 21030 LOCATE 1,XP-1:GOSUB 13ØØØ
IO 21040 RETURN
```

©

Loading And Linking Commodore Programs

Part 5
The Commodore 128

Jim Butterfield, Associate Editor

This month's installment concludes the series by discussing load/link techniques on Commodore's newest eight-bit computer, the 128. As you'll see, the 128's powerful BASIC has simple, built-in commands to perform jobs that require programming tricks on earlier Commodore computers.

There are three major ways to connect programs together. *Chaining* allows several programs to perform a job, each program continuing the work that a previous program has done. *Load linking* enables one program to call another, with the new program starting fresh on a new task. *Overlaying* allows a main program to call in supplementary material, such as machine language subroutines, data tables, or additional screens. All these techniques are easy on the Commodore 128 in 128 mode. For 64 mode, of course, you can use the techniques explained in previous articles in this series.

Chaining

A program that is chained is broken into separate modules, and each part runs separately. The programs may proceed in a specific order; for example, an input program may be followed by a sorting program and then an output program. Or a menu program may call in other programs that you request.

The 128's DLOAD command makes disk chaining extraordinarily easy. If a program executes the statement DLOAD"THISPROG", the computer loads and runs the program named THISPROG. Variables from the earlier program are preserved, so the new program can continue where the old one left off.

The chaining pitfalls of earlier Commodore machines don't apply to the 128. Because the 128 stores the BASIC program text in a different bank of RAM from the working values (variables, strings, and arrays), there is no danger that DLOAD will interfere with variables. The new program simply replaces the old one.

By the way, the 128 has no static strings; all strings, whether static or dynamic, are safely stored in bank 1.

Let's revise the rules for well-chained programs on the 128:

- It doesn't matter whether the first program is bigger or smaller than subsequent programs.
- Strings, variables, and arrays are passed from program to program.
- If you use DEF FN definitions, redefine them in each program module.
- Arrays should be DIMensioned only once, preferably in the first program.

A Short Example

Let's write the first of a small series of Commodore 128 programs which chain together. We'll call our first program START, and it assumes that you want to record grades for eight students.

```
110 PRINT "SIMPLE GRADEBOOK
    DEMO"
120 DIM N$(15),M(15)
130 N=8
140 FOR J=1 TO N
150 PRINT "STUDENT";J;
160 INPUT "NAME";N$(J)
170 INPUT "SCORE";M(J)
180 NEXT J
190 DLOAD "MENU"
```

When the program runs to this point, we have data on eight students. Save the program using the filename START.

Now let's enter the menu program. Type NEW and enter this:

```
100 PRINT
110 PRINT "DO YOU WANT TO--
    "
```

```
120 PRINT "1. CALCULATE AVE
    RAGE SCORE"
130 PRINT "2. CALCULATE HIG
    H/LOW SCORES"
140 PRINT "3. QUIT"
150 PRINT
160 INPUT "YOUR CHOICE (1-3
    )";C
170 ON C GOTO 300,310,320
180 GOTO 160
300 DLOAD "C.AVG"
310 DLOAD "C.HIL"
320 END
```

Note that line 300 won't run into line 310, nor 310 into 320. The moment the program executes DLOAD, the new program loads and begins running. After checking this program closely, save it on disk with the filename MENU. (The name is important; don't substitute any other filename.)

Now type NEW and enter this program:

```
100 PRINT
110 A=0
120 FOR J=1 TO N
130 A=A+M(J)
140 NEXT J
150 PRINT "AVERAGE SCORE FO
    R";N;"STUDENTS =";A/N
160 PRINT
170 DLOAD "MENU"
```

Check this closely and save it as C.AVG. Again, the filename is important. Now type NEW and enter this program:

```
100 PRINT
110 H=M(1):L=M(1)
120 FOR J=1 TO N
130 IF H<M(J) THEN H=M(J)
140 IF L>M(J) THEN L=M(J)
150 NEXT J
160 PRINT "HIGH SCORE WAS";
    H;"BY:"
170 FOR J=1 TO N
180 IF H=M(J) THEN PRINT N$
    (J)
190 NEXT J
200 PRINT "LOW SCORE WAS";L
    ;"BY:"
210 FOR J=1 TO N
220 IF L=M(J) THEN PRINT N$
    (J)
230 NEXT J
240 PRINT
250 DLOAD "MENU"
```

Again, check your typing closely and save it as C.HIL to complete the set.

Now you can experiment with chaining on the 128 by loading the first program (filename START). Note that this program is smaller than both MENU and C.HIL. On earlier Commodore computers, that would be a problem. But it doesn't matter on the 128.

Load Linking

Chaining links one program to the next while keeping the first program's working values intact. That's useful when you're continuing a calculation. But sometimes you'd rather throw away these values, allowing the newly loaded program to start fresh. The RUN command does exactly that. No fuss or bother—just specify the appropriate program name and you're in business. the old variables disappear, the pointers are reset, and the new program starts running.

To illustrate, let's write two very simple programs and use a menu program to select which one to use. Type NEW, then enter this simple square root program:

```
100 PRINT "TABLE OF SQUARE
    {SPACE}ROOTS"
110 FOR J=1 TO 20
120 PRINT J, SQR(J)
130 NEXT J
```

You can try running the program if you want. Save it with the filename SQUARE.

Now type NEW again and enter this simple cube root program:

```
100 PRINT "TABLE OF CUBE RO
    OTS"
110 X=1/3
120 FOR I=1 TO 20
130 PRINT I, I↑X
140 NEXT I
```

Again, you might like to try running the program. Save it with the filename CUBE. Type NEW again and enter this simple 128 loading program:

```
100 DATA SQUARE,CUBE
110 READ A$(1),A$(2)
120 PRINT "WHICH ROOTS DO Y
    OU WANT--"
130 FOR J=1 TO 2
140 PRINT J;A$(J)
150 NEXT J
160 INPUT "WHICH (1 OR 2)";
    N
170 IF N<1 OR N>2 GOTO 120
180 RUN (A$(N))
```

Note the syntax of the RUN command. If you don't specify a drive number, the computer assumes you want drive 0. If you want to run a program on a disk in drive 1, you would add ,D1 to the end of the filename. And if you want to use a variable for the filename (as shown above), it must be enclosed in parentheses.

When you run the menu program, it loads and runs SQUARE or CUBE as selected. When the new program runs, all the old variables are scrapped. The second program starts fresh.

Overlaying

This technique brings in extra material to accompany a BASIC program. It might be a machine language routine, a screen, sprite shapes, or data tables. Whereas chaining and load linking move from one BASIC program to another, overlaying lets the same BASIC program continue with new data in memory.

On the Commodore 128, BASIC 7.0's BLOAD command can bring in the material with no problems. It loads the file, and the BASIC program continues with the next statement. It's quite straightforward, especially compared to the gyrations required on earlier Commodore machines.

However, you must take care not to BLOAD information into the same area of memory occupied by the BASIC program itself (a crash usually results). BLOAD lets you specify a load address, but it's usually convenient to BLOAD a file into the same memory area from which it was saved.

Here's a quick example. First, let's set up a short machine language routine that prints a string of characters. Type NEW and enter this program:

```
100 DATA 0,26
110 DATA 162,65
120 DATA 138
130 DATA 32,210,255
140 DATA 232
150 DATA 201,90
160 DATA 144,247
170 DATA 169,13
180 DATA 76,210,255
190 A=18
200 FOR J=1 TO A
210 READ X
220 T=T+X
230 NEXT J
240 IF T<>2525 THEN STOP
250 RESTORE
260 DOPEN#8,"ML,P",W
270 FOR J=1 TO A
280 READ X
290 PRINT#8,CHR$(X);
300 NEXT J
310 CLOSE 8
```

Be sure that line 290 ends with a semicolon. Then run the program. If it stops at line 240, there's a typing error in one of the DATA statements. Otherwise, it creates a one-block machine language routine on

disk called ML. As we'll see in a moment, this routine prints the alphabet on the screen when called into memory.

Here's our main program, which loads the ML module we just created:

```
100 BANK 15
110 BLOAD "ML"
120 PRINT "HERE IS THE ALPH
    ABET---"
130 SYS 6656
140 PRINT "HERE IT IS AGAIN
    ..."
150 SYS 6656
160 PRINT "THAT'S ALL."
```

BLOAD just brings the ML routine into memory and makes it available to the BASIC program. Simple and effective.

Overlays are popular with machine language programmers on the 128 partly because they are so easy to do and partly because of the mobility of BASIC programs. Depending on recent graphics activities, a BASIC program might start at address 7169 (the usual place) or at 16385 (if a graphics area has been allocated). Rather than puzzle over how to fit a machine language routine into memory with these uncertain locations, many programmers use an overlay. That way they know where the routine loads, even if they're not sure where the BASIC program might be.

Compared to the complexity of earlier Commodore computers, these techniques are a snap on the Commodore 128 in 128 mode. Just remember: DLOAD for chaining, RUN for load linking, and BLOAD for overlaying. ©

Apple ProDOS Catalog Sorter

William J. Coohon

Here's a short utility program that helps you organize your floppy disks by displaying or printing sorted directories. It works on any Apple II-series computer with ProDOS.

Sometimes, locating a particular program or file within a large collection of disks is like searching for a contact lens in a bowl of water—especially when any list of your disk directories will inevitably be outdated and in no particular order. "ProDOS Catalog Sorter" helps you eliminate the confusion by sorting ProDOS directories and listing them on your monitor or printer.

Type in the program below and save a copy before running it for the first time. There are two options to consider when typing Catalog Sorter: date and time stamping, and printer set-up. Stamping the date and time on listings is extremely useful for keeping track of how current (or how old) the information is. If your Apple doesn't have a clock, you may remove certain lines from Catalog Sorter or use the date/time-setting program found in the "Reader's Feedback" column in November, 1985 COMPUTE!. Without the date and time, the program prints zeros. To remove this feature, delete lines 280–320, 345, 445, and 780.

You can also determine how your printer should generate hardcopies of the sorted directories. My directory listings are printed at 17 characters per inch with 8 lines per inch spacing. That way, they can be trimmed down to fit neatly inside a disk envelope or storage case. The printer control characters for an Apple Imagewriter are set up in line 440. If you want to substitute your own printer options, simply alter these codes. If you want your printer to use its defaults, delete line 440 completely. The variable P in line 440 is set to a value of 1 to allow the program to reset the printer options later in line 560, which may also have to be altered for other printer control characters.

Sorting Directories

When you run the program, Catalog Sorter prompts you for drive number 1 or 2 (to exit the program at this stage, simply enter 0). Next, you are asked whether you want

the directories sorted. Type N to disable sorting; any other response sorts your directories in alphabetical order. When might you want to disable sorting? Sometimes programs or files are grouped under directories logically, according to their respective functions. But other files—for instance, monthly financial data—might be organized chronologically. Sorting such files alphabetically would make the grouping less meaningful.

After reading the disk directory, the program asks if you wish to view the listing on your screen or route the output to your printer. It tells you which directory is being sorted (if you choose to sort), then prints the list and moves on to the next directory (if any others exist). Multiple directories are read, sorted, and listed separately to maintain the order of the directory hierarchy. The bottom line of the directory list indicates how many disk blocks are free, how many are used, and the total available (see figure). When all directories on a disk are read, sorted, and listed, the program gives you the option to quit or repeat the process.

If the directory list is displayed on the screen, it appears in the same format as if you typed ProDOS CATALOG command—in 80 columns (note that the abbreviated CAT command uses a 40-column format). This is rather difficult to read on a 40-column display, so an 80-column screen is recommended unless you're interested mainly in the hardcopy listings. Line 260 sets up S$ to switch to the 80-column screen, assuming that the 80-column hardware is addressed at slot 3 (the normal slot for the IIe and IIc). If you wish, you can modify Catalog Sorter to display only 40 columns on each line: Change the PR#3 in line 260 to PR#0.

Apple ProDOS Catalog Sorter

For instructions on entering this listing, please refer to "COMPUTE!'s Guide to Typing In Programs" in this issue of COMPUTE!.

```
2B 100 DIM SK$(100),DA$(100),SD$
       (10)
13 110 GOTO 260
0D 120 S = E
0F 130 S = INT (S / 2): PRINT ".
       ";
CC 140 IF S = 0 THEN 720
FC 150 K = E - S:J = 1
65 160 I = J
34 170 L = I + S
BB 180 IF SK$(I) < = SK$(L) THEN
        230
6F 190 T$ = SK$(I):SK$(I) = SK$(
       L):SK$(L) = T$
F1 200 U$ = DA$(I):DA$(I) = DA$(
       L):DA$(L) = U$
91 210 I = I - S
73 220 IF I > = 1 THEN 170
2C 230 J = J + 1
F0 240 IF J < = K THEN 160
9B 250 GOTO 130
19 260 D$ = CHR$ (4):ES$ = CHR$
       (27):P$ = D$ + "PR#1":S$
       = D$ + "PR#3"
5D 270 P = 0
31 280 D = PEEK (49040) - INT (
       PEEK (49040) / 32) * 32
40 290 Y = INT ( PEEK (49041) /
       2)
7E 300 M = ( PEEK (49041) - Y *
       2) * 8 + INT ( PEEK (4904
       0) / 32)
89 310 MI = PEEK (49042):H = PEE
       K (49043)
AE 320 MI$ = STR$ (MI): IF MI <
       10 THEN MI$ = "0" + MI$
51 330 E = 0:C = 0:F = 0: PRINT
       S$: PRINT
F6 340 HOME : HTAB 32: PRINT "Pr
       oDOS CATALOG SORTER"
6C 345 GOSUB 780
5B 350 VTAB 8: INPUT "Drive <1>
       or <2>, <0> will END? ";N
AD 360 IF N = 0 THEN GOTO 560
92 370 IF N < 1 OR N > 2 THEN 34
       0
43 380 PRINT : INPUT "<S> sort o
       r <N> no sort? ";B$
76 390 IF B$ < > "N" AND B$ < >
       "n" THEN B$ = "S"
EC 400 GOSUB 600: GOSUB 620
E9 410 PRINT : INPUT "<S> screen
        or <P> printer? ";A$
E3 420 IF A$ < > "P" AND A$ < >
       "p" THEN HOME : GOTO 450
AA 430 A$ = "P": PRINT P$
E5 440 IF P = 0 THEN PRINT CHR$
       (9)"136N": PRINT ES$; CHR
       $ (81);ES$; CHR$ (66):P =
        1
6D 445 GOSUB 780
17 450 PRINT : PRINT L1$: PRINT
93 460 PRINT L2$: PRINT L3$: GOS
       UB 730
93 470 IF C = 0 THEN 510
7A 480 FOR ID = 1 TO C
4D 490 GOSUB 600:L1$ = L1$ + SD$
       (ID): GOSUB 620
0F 500 PRINT : GOSUB 580: PRINT
       L1$: GOSUB 730: NEXT
1C 510 GOSUB 580: PRINT LEFT$ (L
       5$,64);"   FILES: "F
C9 520 IF A$ = "P" THEN PRINT S$
5C 530 PRINT : INPUT "MORE (Y,N)
       ? ";A$
31 540 IF A$ = "N" OR A$ = "n" T
       HEN 560
9C 550 GOTO 330
DA 560 IF P = 1 THEN PRINT P$: P
       RINT ES$; CHR$ (99): PRIN
       T S$
8A 570 PRINT "Bye!": END
A5 580 IF A$ = "P" THEN PRINT P$
27 590 RETURN
C8 600 PRINT D$"PREFIX,D"N
D0 610 PRINT D$"PREFIX": INPUT L
       1$: RETURN
37 620 PRINT D$"OPEN "L1$",TDIR"
F4 630 PRINT D$"READ "L1$
F0 640 INPUT L1$:E = 0
AC 650 INPUT L2$: INPUT L3$
E4 660 INPUT L4$
D8 670 IF L4$ = "" THEN 710
46 680 E = E + 1:SK$(E) = MID$ (
       L4$,2,15):DA$(E) = L4$
74 690 IF MID$ (L4$,18,3) = "DIR
       " THEN C = C + 1:SD$(C) =
        MID$ (L4$,2,15)
19 700 GOTO 660
63 710 IF B$ = "S" THEN PRINT "N
       ow sorting "L1$".";: GOTO
        120
4C 720 INPUT L5$: PRINT : PRINT
       D$"CLOSE ": RETURN
9F 730 GOSUB 580:F = F + E
E8 740 IF E > 0 THEN FOR I = 1 T
       O E: PRINT DA$(I): NEXT
7E 750 IF E = 0 THEN PRINT "  (D
       IRECTORY EMPTY)"
A8 760 PRINT : IF A$ = "P" THEN
       PRINT S$
25 770 RETURN
EB 780 HTAB 28: VTAB 4: PRINT "D
       ATE: "M"/"D"/"Y"    TIME:
        "H":"MI$: RETURN
```

©

A sample directory listed generated with "ProDOS Catalog Sorter."

```
DATE: 12/30/85   TIME: 19:00

/CDOHON.300EC85

 NAME            TYPE  BLOCKS  MODIFIED         CREATED          ENDFILE SUBTYPE

*BASIC.SYSTEM     SYS      21  <NO DATE>         8-DEC-85 22:00   10240
 CAT.SORT         BAS       5  30-DEC-85 18:30  30-DEC-85 18:30    1566
 CAT.SORT.COPY2   BAS       5  30-DEC-85 18:30  30-DEC-85 18:30    1566
 CAT.SORT.TEXT    TXT       8  30-DEC-85 19:00  30-DEC-85 19:00    3546 R=     0
*PRODOS           SYS      31  <NO DATE>         8-DEC-85 22:00   15360

BLOCKS FREE:  203    BLOCKS USED:   77    TOTAL BLOCKS:  280    FILES: 5
```

Mandelbrot Graphics For Commodore

Steven M. Thorpe

A mathematics phenomenon known as the Mandelbrot set can provide the basis for some stunning computer graphics. The following programs make it possible to generate a wide variety of colorful high-resolution images which can be saved on disk for future viewing. The programs work on any Commodore 64 (or 128 in 64 mode).

One of the most beautiful features of the Commodore 64 is its multicolor bitmap mode, which lets you create detailed high-resolution images using several colors. We're all familiar with the many drawing programs that work like video paint boxes, letting you draw directly on the screen. But the computer can create beautiful, highly intricate images all by itself, using relatively simple mathematical methods. "Mandelbrot Graphics For Commodore" allows you to generate interesting hi-res pictures, save them to disk, and reload them at any time.

Creating Screens

Type in and save Program 1. When you run the program, it immediately clears the hi-res screen and begins to draw an image based on the *Mandelbrot set* (see below). You'll need to be patient: A full-screen hi-res image takes a long time to create. Although the program uses machine language routines to clear the hi-res screen, the drawing computations are done in BASIC.

In multicolor bitmap mode, up to four different colors can be displayed in each character position. Distortion occurs if a program calls for too many colors in a single position. Since you can have only one screen background color at a time, each character position is actually limited to three independent colors. While this may seem a severe restriction, spectacular graphics are still possible. Program 1 selects an available color memory source (not yet used in the current character position unless used for the current color), and then sets the appropriate color code and bit pattern to display that color.

If you're impatient for results and don't mind viewing a smaller image, replace lines 180 and 190 with these lines (be sure you've saved the original version of the program before you make this modification):

```
180 FOR X0=XL TO XR STEP (XR-XL)/
    159*2
190 FOR Y=YT TO YB STEP (YB-YT)/
    199*2
```

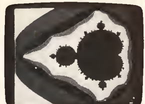
With this change, Program 1 draws an image about one-fourth the size of the screen, in roughly one-fourth the time it would take to draw a full-screen picture. To change back to a full screen, retype lines 180 and 190 as listed in Program 1. This can be useful when modifying the program to produce different results. You can view the results in a reduced scale to save time, then draw it at normal size to be saved to disk. Press RUN/STOP–RESTORE if you need to break out of the program.

The variables XL and YB play a crucial part in defining what the final image looks like. XL sets the left boundary of the Mandelbrot set and YB sets the bottom. Similarly, XR defines the right boundary and YT defines the top. By changing the values of XL, YB, and SR, you can "zoom in" for a closer look at any given area of the Mandelbrot set. You don't need to worry about the value of YT: The program automatically gives YT the value needed to shape the screen image correctly.

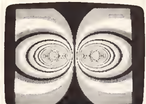
Line 170 contains color codes used for various parts of the image. Changing these numbers alters the colors used in the display (see your user's manual for an explanation of color codes). Lines 222–230 shape the zones for each different color. In line 120, the variable SM determines the spacing between different colors, and CT controls how much detail is shown. Remember, using too many colors in zones of rapid color change can lead to excessive color distortion. Together, the variables SM and CT affect how long it takes to complete a Mandelbrot image.

Invisible Lines Of Power

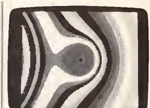
Programs 2 and 3 contain modifications for Program 1 which generate different displays based on the unseen forces of nature. To use these programs, you must already have a copy of Program 1. To enter Program 2, first load Program 1 into memory, then type in the lines listed in Program 2 (they will replace lines 100–240 of the original



"Mandelbrot Graphics For Commodore" creates detailed hi-res images based on an unusual mathematics technique.



This image simulates the lines of electromagnetic force surrounding two electrical conductors.



This pattern is created by a program that represents the gravitational forces around the earth and the moon.

program). When that's done, save the program using a different filename from the one you used for Program 1.

Program 2 uses the color schedule to paint patterns that resemble the magnetic lines of force between two wires charged with electricity. Displays of this type make it easier to visualize forces such as electromagnetism, which are invisible to the naked eye, yet have a profound effect in the world around us. Of course, you can enjoy the displays even if you don't understand the scientific concepts involved in these simulations.

You can also modify Program 2 to produce different effects. Change the values of variables I1 and I2 to see what the magnetic pattern is like for various current levels. For instance, if you set I1 to 25 and I2 to −100, the program displays an opposed current flow with one current four times as large as the other. Larger current levels cause closer flux lines, more detail, and a greater chance of color distortion.

To enter Program 3, load Program 1, type the lines listed for Program 3, then save the entire program under a unique filename. Program 3 simulates another invisible, yet powerful force of nature—the interaction of gravitational forces between the earth and the moon. The black region on the right is the zone of gravity equal to that at the surface of the moon. The program accurately draws the moon to scale relative to the gravitational potentials in effect.

Line 120 of Program 3 positions the earth and moon relative to the upper left-hand corner of the screen, using an *x,y* coordinate scheme in which the screen has horizontal positions 0–159 and vertical positions 0–199. The earth can be shown by assigning it a value on the screen. You can create an interesting image by placing the earth at coordinates 53,99 and the moon at coordinates 106,99.

The variable F determines which color is plotted. F represents the gravitational force (measured in Newtons) on a stationary one-kilogram mass. Variable DE represents the distance from earth, and DM signifies the distance from the moon. The region close to the moon was made one color because it has such a steep gravitational gradient that there would be excessive color distortion if it were not specially treated.

Saving And Loading Screens

After you create a screen, it's easy to save it on disk for future use. Simply press S after the image is completely drawn. Before saving an image, make sure that the disk drive contains a disk with at least 40 blocks free. Three different memory areas must be saved in separate disk files to reconstruct the image accurately.

To avoid spoiling the image with a printed prompt, the program switches temporarily to a text screen to allow you to enter a filename (which should contain no more than 12 characters). The program then manipulates the name you enter to create distinct filenames for the three files necessary to store all the picture information. The prefix *M–* is added to the filename you specified for the first file. This file contains the hi-res bitmap of the image. For the next file, the suffix *S* is added in addition to the *M–* prefix. This file contains the screen memory for the image. Finally, for the third file, the suffix *SC* is added in addition to the *M–* prefix. This file contains the color memory for the image.

Choose your filenames carefully; the program erases any existing files with the specified filename before it stores the new files. Thus, if you accidentally reuse an existing name, you may overwrite a previously saved masterpiece.

After you enter the filename, the program switches back to the hi-res image while the screen is being saved.

Program 4 loads the stored image from disk and displays it on the screen. When you run this program, it asks you to enter a filename. Give the filename you specified when you saved the image with Programs 1–3. Don't worry about the prefixes and suffixes; Program 4 takes care of these and loads all three files necessary to recreate the image.

What Is A Mandelbrot Set?

Although Mandelbrot images have attracted wide interest as a means of generating graphics, the origin of these pictures lies deep in the realms of mathematics and engineering. In brief, certain engineering problems require the use of *complex numbers.* A complex number consists of two parts: a *real* part and an *imaginary* part. The real part of the number is like the numbers we use every day. For the imaginary part, special rules of multiplication apply. For instance, if you square the imaginary part of a complex number, the result becomes negative and real. Thus, imaginary numbers are often written in engineering calculations as a real

number multiplied by the constant *i*, where *i* can be considered the square root of −1.

The real and imaginary parts of a complex number can be plotted on a coordinate grid known as the *complex plane*. Benoit B. Mandelbrot, a researcher at IBM, discovered that points inside a certain region of the complex plane behave strangely when they are repeatedly squared and the result of each squaring is added to the original point. Some points get large rapidly, while their near neighbors grow slowly. Other neighboring points don't grow at all. As the process is repeated, these tendencies are accentuated. The region where these strange results occur is called the *Mandelbrot set* after its discoverer.

While Mandelbrot sets have certain practical applications, they can produce striking results when you translate the numeric values into different-colored points on a computer screen. These programs color each point on the hi-res screen according to how fast it grows during the iterative (repetitive) process. Computing the entire set requires an enormous number of calculations, so all of these programs take some time to complete—many hours in some cases. You may want to start a program running at bedtime so the picture will be finished by the next morning. (There's no need to leave the monitor on all night, of course.)

If you'd like to photograph a finished image, shoot the photos in a darkened room to eliminate screen glare. Load a single-lens reflex camera with medium-speed film (ISO 100 works well), mount the camera on a tripod, attach a cable release, aim the viewfinder squarely at the screen, and use a very slow shutter speed—no faster than 1/4 second. If your camera has an automatic exposure mode or a built-in light meter, it should indicate a lens aperture of f/8 to f/16.

For instructions on entering these listings, please refer to "COMPUTE!'s Guide to Typing In Programs" in this issue of COMPUTE!

Program 1: Mandelbrot Graphics

```
QR 10 POKE56,32:POKE55,0:CLR:FORA=828TO908:READB:POKEA,B:NEXT
RR 20 DATA 162,2,160,4,208,4,162,4,160,2,142,101,3,142,127,3,140,103,3
GD 30 DATA 140,129,3,169,0,133,2,133,4,169,206,133,3,169,200,133,5,162,3
AE 40 DATA 160,0,177,2,145,4,136,208,249,230,3,230,5,202,16,242,169,4,133
JF 50 DATA 3,169,204,133,5,162,3,160,0,177,2,145,4,136,208,249,230,3,230
EK 60 DATA 5,202,16,242,96
AB 100 REM ------ MANDELZOOM-64 ------
FH 120 SM=7:CT=40:REM DETERMINES DETAIL
CE 125 REM DEFINE REGION EXAMINED BELOW
GK 130 XL=-2.2:XR=.58:YB=-1.25
BF 137 DX=(XR-XL):YT=YB+DX*.9
MS 140 DIMCO(15):FORI=0TO15:READCO(I):NEXT
AC 150 GOSUB360:M=159/DX:B1=-M*XL
BJ 160 MM=199/(YB-YT):BB=-MM*YT
MK 170 DATA 1,3,4,2,8,7,13,5,14,6,4,15,14,6,4,15:REM{SPACE}<-- COLOR CODE SCHEDULE
DS 180 FOR X0=XL TO XR STEP(XR-XL)/159
PQ 190 FOR{2 SPACES}Y=YT TO YB STEP(YB-YT)/199
EF 200 A=X0*X0-Y*Y+X0:B=2*X0*Y+Y:C=0
RJ 210 R=A*A-B*B+X0:I=2*A*B+Y:C=C+1:S=USR(R*R+I*I):A=R:B=I:IFS<SMTHENIFC<CTTHEN210
QR 220 X1=INT(M*X0+B1+.5)
RQ 222 Y1=INT(MM*Y+BB+.5):IFC<6THENCO=CO((C/15-INT(C/15))*15):GOTO240
QF 225 IFC<CTTHENCO=1:GOTO240
XK 230 IFC=CTTHENCO=0
SB 240 GOSUB440:NEXT Y,X0
DF 250 REM SAVE PICTURE IF 'S' IS PRESSED
KX 260 GETA$:IFA$<>"S"THEN260
FB 261 SYS828:POKE53272,21:POKE53265,27:POKE53270,200
SF 262 PRINT"{CLR}{2 DOWN}ENTER FILENAME (UP TO 12 CHAR)":INPUTF$:F$="M-"+LEFT$(F$,12)
8Q 270 POKE53272,29:POKE53265,59:POKE53270,216:SYS834:K=8192:E=16191:GOSUB300
EM 280 F$=T$+"S":K=1023:E=2023:GOSUB300
XH 290 F$=T$+"C":K=55296:E=56295:GOSUB300:GOTO260
EM 300 F$="0:"+F$:OPEN15,8,15,"S"+F$:CLOSE15:T$=F$
SJ 310 ZK=PEEK(53)+256*PEEK(54)-LEN(T$):POKE782,ZK/256:POKE781,ZK-PEEK(782)*256
HM 320 POKE780,LEN(T$):SYS65469:POKE780,1:POKE781,8:POKE782,1:SYS65466
EE 330 POKE254,K/256:POKE253,K-PEEK(254)*256:POKE780,253:K=E+1:POKE782,K/256
8D 340 POKE781,K-PEEK(782)*256:SYS65496:T$=RIGHT$(T$,LEN(T$)-2):RETURN
BX 350 REM----- SET COLOR GRAPHICS MODE AND CLEAR MEMORY TO BE USED -----
XF 360 PRINT"{CLR}":FORI=49152TO49248:READJ:POKEI,J:NEXT:POKE53280,0:POKE53281,0
CJ 370 POKE251,0:POKE252,32:POKE253,63:POKE254,63:POKE49169,0:SYS49166
AK 380 SYS49152:POKE785,39:POKE786,192
XD 390 BASE=8192:POKE53272,PEEK(53272)OR8
G8 400 POKE53265,PEEK(53265)OR32:POKE53270,PEEK(53270)OR16
HM 410 POKE251,0:POKE252,216:POKE253,231:POKE254,219:SYS49166:RETURN
RQ 420 REM ---- PLOT X1,Y1,CO{SPACE}------------
AH 430 REM{2 SPACES}0<=X1<160{2 SPACES}0<=Y1<200 0<=CO<15
AJ 440 IF CO=0 THEN RETURN:REM USE SCREEN COLOR CODE{SPACE}IN 53281
EJ 450 X=2*X1+1:RO=INT(Y1/8):CH=INT(X/8):LN=Y1AND7:BY=BA+RO*320+8*CH+LN
SJ 460 REM --- SET COLOR AND PIXEL ---
RG 470 SB=1024+RO*40+CH:REM SCREEN BYTE
EF 480 SE=PEEK(SB):S9=SEAND240
JK 490 REM USE HI NYBBLE OF SB IF OK
KC 500 IF(S9=0ORS9=CO*16)THEN{SPACE}POKE SB,SEOR(CO*16):GOTO580
GF 510 REM USE LO NYBBLE OF SB IF OK
XR 520 S8=SEAND15:IF(S8=0ORS8=CO)THEN POKE SB,SEORCO:X=X-1:GOTO580
SG 530 REM USE COLOR MEM. IF IT IS OK
KJ 540 CM=SB+54272:REM COLOR NYBBLE ADDR.
PD 550 C1=PEEK(CM):C3=C1AND15
GF 560 IF(C3=0ORC3=CO)THENPOKECM,C1 ORCO:GOSUB580:X=X-1
EB 570 REM TOO MANY COLORS THIS CHAR.
AE 580 BI=7-(XAND7):POKEBY,PEEK(BY)OR(2↑BI):RETURN
HD 590 DATA 169,0,133,251,133,253,169,4,133,252,169,8,133,254,160,0,169,27,145,251
ES 600 DATA 230,251,208,2,230,252,165,251,197,253,208,240,165,252,197,254,208,234
DH 610 DATA 96,32,43,188,240,52,16,3,76,72,178,32,199,187,165,97,56,233,129,8,74
DF 620 DATA 24,105,1,40,144,2,105,127,133,97,169,4,133,103,32,202,187,169,92,160,0
FJ 630 DATA 32,15,187,169,87,160,0,32,103,184,198,97,198,103,208,233,96
```

Program 2: Magnetic Forces

```
HX 100 REM------- MAGNETIC FIELD -------
CR 120 DIMCO(15):FORI=0TO15:READCO(I):NEXT
```

```
SG 130 REM WIRE LOCATIONS, CUR
       RENT LEVELS & PHYSICAL
       {SFACE)CONSTANTS
MM 140 XA=53:YA=99:XB=107:YB=9
       9:I1=100:I2=-100:K=2E-7
       :GOSUB360
GR 150 DATA 0,0,0,6,6,6,6,5,5,
       5,5,5,4,4,4,4:REM <----
        COLOR CODE SCHEDULE
DC 160 FOR X0=0 TO 159 : FOR Y
       =100 TO 199
PG 170 DA=UBR((XA-X0)↑2+(YA-Y)
       ↑2)
0F 100 DB=USR((XB-X0)↑2+(YB-Y)
       ↑2)
FD 190 B=K*I1/DA+K*I2/D0
QJ 200 X1=X0:Y1=Y:CO=ABB(INT(B
       *1E0+.5)):CO=CO({CO/15-
       INT(CO/15))*15):GOSU042
       0
GP 210 Y1=199-Y:GOSUB440:NEXT
       {SPACE}Y,X0
HH 220 REM
PH 230 REM
XJ 240 REM
```

Program 3: Forces Of Gravity

```
0R 5 REM----- EARTH-MOON SYSTE
     M-----
PD 100 DIMCO(15):FORI=0TO15:RE
       ADCO(I):NEXT
RR 110 REM BODY LOCATIONS & MA
       BSES
GC 120 XE=-111:YE=99:XM=80:YM=
       99:EM=6E24:MM=7.3E22:G=
       .667E-10:GOBUB360:XL=XE
       :Y0=YM
KM 130 CF=3.8E0/(XM-XE)
EA 140 DATA14,7,11,2,9,0,7,13,
       5,3,14,6,4,2,9,8:REM <-
       --- COLOR CODE SCHEDULE
BB 150 REM CF CONVERTS 0CREEN
       {SPACE}UNITS TO METERS
       {0PACE)AND 1.40 CORRECT
       S X-Y ASPECT
RX 160 FORX0=0TO159:FORY=0TO99
EX 170 D1=XE-X0:D2=(YE-Y)/1.48
       :D3=XM-X0:D4=(YM-Y)/1.4
       8
FM 100 DE=U0R(D1*D1+D2*D2)*CF+
       1:DM=USR(D3*D3+D4*D4)*C
       F+1
XD 190 FE=G*EM/DE/DE:FM=G*MM/D
       M/DM:FX=FE*COS(D1/DE)+F
       M*COS(D3/DM)
KQ 200 FY=FE*SIN{D2/DE)+FM*0IN
       (D4/DM)
FM 210 F=USR(FX*FX+FY*FY)
QH 220 X1=X0:Y1=Y:CO=INT(F*5E3
       +.5):CO=CO(ABS((CO/15-I
       NT(CO/15))*15))
XH 222 REM
DJ 225 REM
SF 230 IFF>.0055THENIFX0>40THE
       NCO=4:IFF>.167THENNEXTX
       0
FH 240 GOSUB450:Y1=199-Y:GOSUB
       450:NEXT Y,X0
```

Program 4: Image Loader

```
PS 100 REM---MULTI-COLOR BIT M
       AP RECALL---
HF 110 INPUT"{CLR}{3 DOWN)ENTE
       R FILENAME";F$:F$="M-"+
       F$
FG 120 FRINT"{CLR}":FOKE53272,
       (FEEK(53272)AND240)OR0:
       FOKE53265,FEEK(53265)OR
       32
CR 130 FOKE53270,PEEK(53270)OR
       16:FOKE53200,0:POKE5320
       1,0
SR 140 F$="0:"+F$:GOSU0180
AR 150 F$=F$+"B":GOSUB100
EH 160 F$=F$+"C":GOSUB180
FX 170 GOTO170
AM 100 T$=F$:ZK=FEEK(53)+256*P
       EEK(54)-LEN(T$):FOKE702
       ,ZK/256
MJ 190 FOKE701,ZK-FEEK(702)*25
       6:POKE700,LEN(T$):SYS65
       469
EF 200 FOKE700,1:FOKE701,0:FOK
       E782,1:SYS65466
SQ 210 POKE780,0:SYS65493:RETU
       RN
```

©

Fred D'Ignazio, Associate Editor

It Only Takes Two To Make Music

For 20 years Paul Lehrman has dreamt about becoming a composer. "But," he says, "serious composers often end up starving to death."

Paul has a lot of experience in the music industry—as a musician, recording engineer, and most recently as vice president of Southworth Systems, the makers of *Total Music*, a MIDI music program for the Apple Macintosh. "I'm not a serious composer," he says. "I'm a pop guy. But I want to help serious composers make music."

Paul is convinced that computers can help composers. To prove this, he recently did what no one has done before: He created an entire album of music with a personal computer. His album, *The Celtic Macintosh*, consists of traditional and contemporary Irish and Scottish jigs, reels, hornpipes, airs, and laments. It took him less than three weeks, start to finish, and cost him a little under $100. (He had to buy five floppy disks, a VHS videotape, and rent a digital tape converter.)

Why Irish and Scottish folk songs? St. Patrick's Day was coming, and Paul's Irish-American friend Sharon Kennedy, a storyteller, asked him to put together "something quick" that she could play over the PA system as background music at the annual St. Patrick's Day Concert in Brookline, Massachusetts. She was so happy with the result that Paul decided to make a whole album and sell it at the concert.

A Home Studio

Paul's recording studio was in his living room. Each morning he would get up at 10:00 a.m., shower, eat breakfast, walk the eight steps to his studio, and go to work—in jeans and a bathrobe. His instruments consisted of three synthesizers (a Kurzweil K250, a Yamaha DX7, and a Casio CZ-101), a drum machine (Roland PR-707), a digital effects processor (Lexicon PCM70), a keyboard mixer, and the Fat Mac running Southworth's *Total Music* software.

Paul did not look especially historic or impressive sitting alone in a bathrobe in his living room typing on a Mac and flanked by a few keyboards. But looks are deceiving. To really appreciate what he has done, you have to listen to his album. You hear woodwinds, an accordion, a hammer dulcimer, guitars, an acoustic piano, penny whistles, drums, trumpets, a harp, flutes, tympani, and many other instruments—some of which he invented. And, thanks to the PCM70 (which doubles as a reverberator), the songs sound as if they were played in all sorts of places—from a small, cozy bar to a grand concert hall. I was totally fooled. It is a masterful audio illusion. But it is also fine music.

Using the Mac, Paul called up instrument sounds on each of the keyboards, played them on the Kurzweil (which acted as his master keyboard), and recorded them as a single track on a 3½-inch floppy disk. He replayed the tracks, polished them, then overdubbed new tracks on top of the old. By layering the tracks together, he created the illusion of an entire band or orchestra.

With the press of a button, the computer played all the instruments while Paul taped them on a VHS video recorder. To make the recording sound professional, he used a Sony PCM (pulse code modulation) converter he rented for $10 a day to convert the analog sound signal into a digital signal. When recorded on the videotape, this digital audio equals the sound quality of a compact disc.

Next he went to a local record duplication house, which copied his master tape onto standard audio cassettes—50 at a time.

Not Machine Music

"People expect music made by machines to sound like machine music," says Paul. "I made this album to disprove that. I used the computer to do the things it does well, and made it play the kind of music I wanted to play. I deliberately left in things that some people might call mistakes—little timing things and grace notes. I could have fixed them, but I didn't. I wanted the music to sound like it was made by a human being. Without that human element the music becomes rhythmically perfect all the time. It sounds boring, robotic.

"I wanted to show that a composer could set up everything in his living room and not spend a lot of money. He can create and record his own music all on his own. A system like this cuts all the complications and red tape separating a composer from his audience. Now there is no one between them.

"When I was a kid I went to see Mary Martin do a concert in New York City. I remember she sang a song called 'It Takes Three to Make Music'—one to write, one to play, one to hear. That equation is changing. Now, in a very real sense, the person who writes and the person who plays can be the same person. So it only takes two to make music. And we're not talking about playing a piano, either. We're talking about having an entire orchestra at your disposal. Any kind of orchestra, with any kind of instruments."

To learn more about Paul's system, write to Paul D. Lehrman, 31 Maple Ave. Apt. #1, Cambridge, MA 02139. To get a tape of *The Celtic Macintosh*, send $10. To learn more about *Total Music*, write Southworth Music Systems, Inc., Box 275, RD 1, Harvard, MA 01451. ©

Metaphorical Computing

Every time we use a computer, we are working with metaphors. In some cases the metaphor is obvious, and in others it is not. A game, for example, might provide us with a model of a city through which we have to navigate without being caught by the bad guys. Rationally, we know there is no city or bad guys, but this metaphor lets us move beyond the computer to the application itself—the computer becomes the mirror of the mind's eye.

This aspect of the computer is what makes it so useful. Metaphors like desktops, menus, and windows can make computing appear more like "real" activities. Those who prefer command line interpreters to desktops are merely expressing a preference for one metaphor over another. None of this matters to the machine. As far as the computer is concerned, code is code and bytes are bytes.

High-level computer languages are programs and are therefore metaphors as well. As the noted computer scientist Edsger Dijkstra says in his book *A Discipline of Programming*: "There are two machines—the physical machine—the one that you pay for, that breaks down, that takes space on your desk. Then there is the abstract machine—defined by the functions the machine is to perform."

As programmers, we trust that the abstract machine is implemented in the physical machine. In fact, it might not be. For example, a program could accept 6502 machine language and convert it to run on a 68000 microprocessor. From the programmer's perspective, the computer is a 6502. The point here is that, even at the most basic level of programming, one cannot escape from metaphors.

Generations Of Languages

If computation is metaphorical at the machine level, it becomes even more so as we move to higher-level languages. In assembly language, for instance, the metaphor of the machine is often expanded to include instructions that the computer cannot execute directly. Programmers call these *macros*, or macro instructions. A macro that simulates a missing multiplication command greatly simplifies the programmer's task.

At the first two levels of computer language (machine and assembly), the metaphor pertains to the machine itself. At this level it is the program's purpose to instruct our computers. Once we move to high-level languages like BASIC, FORTRAN, COBOL, LISP, PROLOG, Forth, and Smalltalk, it becomes the computer's task to execute our programs. This may seem like a subtle distinction, but as we shall see, it is not.

I have come to believe that the major differences between various high-level languages are not to be found in their syntaxes, grammars, or vocabularies, but in their metaphors.

A Pascal programmer, for example, sees a program as a collection of procedures, each of which produces some effect. A LISP programmer sees a program as a collection of functions that output results in much the same manner as mathematical operations. A PROLOG programmer sees a program as a collection of assertions and rules that the computer can use as needed to respond to a query.

Now we can begin to see why it's so difficult to write major application programs. The programmer has the task of creating one metaphor out of another one. This is why multilingual programmers often spend some time deciding which language to use before writing any code.

The trick to making programming easy is to make the metaphor of the language match that of the application.

Construction Sets

Some languages do this quite well. The ease with which a user can create new pinball games with *Pinball Construction Set* is a result of the program's use of the same metaphor, whether a game is being constructed or played.

Other construction set languages (including *Loderunner* and any good spreadsheet program) have become immensely popular. Their popularity arises from ease of use, and this is a direct consequence of the consistency of the metaphor as one moves across the boundary from programming to executing a program.

While construction sets have great value, they also have a strong limitation. They are not general-purpose programming tools. One cannot build a word processor using *Pinball Construction Set* or a music program with *Multiplan*.

The closest we've come to general-purpose direct-manipulation languages is with products like *Filevision*, a Macintosh-based visual database program. I've used *Filevision* since 1984 as a tool for creating instructional software. But even *Filevision* (which was never intended to be a language) has its limitations.

The key to constructing a general-purpose direct-manipulation language is to find a "metametaphor"—a metaphor that encompasses all application metaphors. Does such a metaphor exist? This question is being asked by those of us who are exploring direct-manipulation programming. So far, the answer is not clear.

In the meantime, *metaphors be with you.* ©

The Beginners Page

Tom R. Halfhill, Editor

Turning Apples Into Oranges

Sometimes it seems as if BASIC is overpopulated with a nearly endless collection of string-handling functions. We've already looked at LEFT$, MID$, RIGHT$, CHR$, ASC, and LEN, not to mention operators for adding strings together and making logical comparisons. Yet, we've barely scratched the surface of what can be done with strings in BASIC. The reason for all these functions is that programmers keep demanding more and more flexibility for manipulating text in their programs. One of BASIC's early predecessors, a language called FORTRAN, was very strong on mathematical functions, but rather weak when dealing with strings. From the very beginning, BASIC has sought to improve in this area.

Two interesting string functions we haven't covered so far are VAL and STR$. These two functions, common to virtually all BASICs, are opposites of each other. STR$ lets you convert a number into a string, and VAL lets you convert a string into a number. For example, look at this program:

```
10 DIM A$(10):REM This line for Atari
   only
20 A$="123"
30 PRINT A$
40 A=VAL(A$)
50 PRINT A
60 A$=STR$(A)
70 PRINT A$
```

When you run this program, the result is:

```
123
123
123
```

So what? you say. We could get the same result by deleting lines 30–70 and simply substituting PRINT A$: PRINT A$:PRINT A$.

But the result would *not* be the same. Although PRINT A$ in line 30 prints the number 123 on the screen, it's printing the number as a *character string*, not a *numeric value*. You can't tell one from another when they're printed on the screen, but the difference is important and matters very much to BASIC. If you attempt a statement such as A=A+"123" or A$=A$+123, BASIC reports some sort of type-mismatch error. Character strings and numeric values just don't mix.

That's why there's VAL and STR$. As shown in line 40, A=VAL(A$) takes the number "123" stored as a *character string* in A$, converts it into the corresponding *numeric value* 123, and stores it in the numeric variable A. Conversely, the statement A$=STR$(A) in line 60 takes the number stored as *numeric value* 123 in A, converts it into the *character string* "123," and stores it in the string variable A$. In short, VAL and STR$ let you turn apples into oranges and back again.

Easier Input

An interesting trick, you say, but what's the purpose? True, VAL and STR$ aren't exactly the most heavily used functions in the BASIC language. Instead, they're like Allen wrenches—once a year when you need them, nothing else will do.

For instance, recently I wrote a short program to experiment with a new computer's sound capabilities. The program asks the user to enter a phone number, then dials the number by playing Touch-Tones through the monitor speaker. To determine which Touch-Tone frequencies to play, I needed to extract each digit in the phone number as numeric data. But to make it as easy as possible for the user to enter the phone number, I wanted the program to accept the keyboard input as a character string. That way, people can type in the phone number any way they want: (919) 555-1212, or 919-555-1212, or 919 555 1212, etc. A statement such as INPUT A$ accepts all those variations. But if the program used INPUT A, the phone number would have to be entered as 9195551212, or an error would result.

So, VAL comes to the rescue. Once the phone number is entered in A$, the program loops through the string, converting digits from 0 to 9 into numbers with VAL. The numbers are then passed along to SOUND statements which play the tones. Spaces, hyphens, parentheses, and other characters are ignored. Here's a simplified version of the routine:

```
100 FOR N=1 TO LEN(A$)
110 IF ASC(MID$(A$,N,1))<48 THEN 150
120 IF ASC(MID$(A$,N,1))>57 THEN 150
130 A=VAL(MID$(A$,N,1))
140 REM SOUND statements to play
    Touch-Tones here...
150 NEXT N
```

This routine conveniently demonstrates several string-handling techniques we've covered in the past few columns. Line 100 sets up a FOR-NEXT loop by using the LEN function to measure the length of A$; this determines how many times the loop repeats (one loop for each character in A$). Lines 110 and 120 use the MID$ function to examine one character at a time in A$; if the ASC function discovers that the character is less than the ASCII value of 48 (the number 0) or greater than the ASCII value of 57 (the number 9), the program skips to line 150 and makes another pass through the loop.

If a character in A$ is a number 0–9, the program falls through to line 130. Here, the VAL function converts the character into a numeric value and stores it in A. Line 140 is where the SOUND statements to play the tones would be inserted. Line 150 then loops back to examine another character in A$.

You can adapt this trick to many of your own programs. Whenever you'd like to accept keyboard input as a string for maximum flexibility, just convert the string with VAL to the numeric input you're really looking for. ©

Arlan R. Levitan

Electronic Bulletin Boards: A Retrospective

Bulletin boards have been with us in one form or another for hundreds of years and will likely stay with us well into the future. Why? What's so special about bulletin boards, electronic or otherwise?

It's difficult to pinpoint when the first bulletin board appeared. Perhaps cave paintings were primitive bulletin boards. In the modern sense of a community communications media, the earliest bulletin board may have been the medieval practice of posting royal proclamations in the center of commerce, the town square.

The traditional bulletin board, with a wide variety of messages tacked to a freely accessible surface, abounds in our supermarkets, factories, offices, schools, laundromats, community centers, and city halls. These bulletin boards are more than just a way to give away kittens or sell tires. They make it possible for people with a message to reach out to the community as a whole.

Electronic Thumbtacks

The thousands of computer-based bulletin board systems (BBSs) which are online today offer the traditional message posting and a great deal more. Imagine trying to maintain a series of communications with other people using a regular bulletin board at a supermarket. Driving to the store every time you want to leave or read a message makes extended communication via corkboard and notecard extremely inconvenient. Even if you make the trip regularly, a less than careful search of the posted messages may miss the very reply that was sought.

The fact that a BBS can be accessed remotely, without leaving one's home, makes an ongoing dialog between many parties a simple matter. A computer dedicated to running the BBS manages the messages; in addition to numbering and indexing the messages, it also automatically notifies its many users of messages intended specifically for them.

The first BBS was born of necessity in 1978. Microcomputers were just getting off the ground, and the first micronauts were few and far between. The four major enclaves of personal computing were located in California, Illinois, Texas, and Massachusetts. Although the computer clubs in these areas exchanged newsletters regularly, there was a decided lack of spontaneous interaction between the major groups and even within the groups themselves.

Ward Christensen and Randy Suess, both members of the Chicago Area Computer Hobbyist Exchange (CACHE), came up with the answer. They developed a program to run on a computer that was equipped with a modem hooked up to a phone line. The program turned the computer into an automated message system. Callers to the Computerized Bulletin Board System (or CBBS, as its originators referred to it) could leave and retrieve messages at any time of day. The CBBS was a huge success, and other clubs began pressing personal computers into service as bulletin boards.

The Spread Of BBSs

CBBS was not a universal program. It was written for computers which used the CP/M operating system (Control Program for Microcomputers). Christensen and Suess wrote a widely publicized article describing the program and the structure of their system as it appeared to the person calling into the CBBS. Realizing that similar programs would be written for other types of computers, they proposed that the functions and commands used by the CBBS be standardized for all BBSs. This would make it unnecessary for people to learn a whole new set of commands for each type of board they accessed.

Sure enough, BBS software for other popular systems soon followed. Craig Vaughn and Bill Blue created a program for Apple II computers called the People's Message System (PMS). Close on their heels was Bill Abney, who produced Forum 80 for the Radio Shack TRS-80, and Tom Giese, father of the Atari Message & Information System (AMIS) for the Atari 400 and 800. Late in 1982, the first version of the Remote Bulletin Board System (RBBS) for the IBM was written by D. Thomas Mack and Jon Martin.

Aside from a message exchange, most BBSs offer a selection of public domain programs and other types of files. By using terminal software capable of receiving files via modem from a remote computer, callers can transfer (download) copies of these files from the BBS to their own machines.

Most of the free software available from BBSs consists of programs that computer enthusiasts like yourself have written and wish to share with other people. A plethora of games, word processors, spreadsheets, database managers, and terminal programs are available for the price of a phone call. Whatever your needs, you can acquire a respectable library of almost-free software that will handle all but the most demanding tasks.

Next Month: Current Trends in Bulletin Board Systems. ©

INSIGHT: Atari

Bill Wilkinson

Tried And True Tools

In keeping with COMPUTE!'s programming languages theme for this month, I'd like to share some thoughts about programming in general and better use of the available languages in particular. I have long contended that, for most purposes, owners of Atari 400/800, XL, and XE computers have all the languages they need. You won't do parallel array processing with a 6502, no matter what language you use, but you *can* balance your checkbook, keep track of your mailing list, access online services via modem, write a book or two, and (of course) play some games. All of those applications and many more have been written with languages now available for the eight-bit Atari computers. What more can you ask for?

In a previous column I said it would be hard for most users to justify trading up to an ST, an Amiga, or whatever. If anything, I feel more strongly about that now. I still write this column using a good old Atari 1200XL (I like its keyboard best) and an Atari 825 printer (ancient history). Sometimes I wish for an 80-column screen or a hard disk drive—keeping track of 200 floppies is not my idea of fun—but I can't justify the expense for the extra convenience.

The same is true when it comes to programming languages. Admittedly, I'm a language junkie. I love learning new languages and/or tricks with old languages. So it would seem that the ST would be a dream machine for me. Despite its youth, the number of languages either available or coming soon is phenomenal: several varieties of BASIC, Logo, Pascal, C, LISP, Modula-2, COBOL, FORTRAN, Prolog, Forth, and 68000 machine language. There are probably others, too.

Old Machines, New Projects

But for owners of eight-bit Ataris, the situation is far from bleak. Though some of the language implementations are not as rich as those on the ST, we can enjoy Pascal, C, Logo, Action!, Forth, PILOT, 6502 machine language, and some extraordinarily easy-to-use BASICs.

Even though I've been using Ataris for six years now, I still see some interesting projects to do—projects that I've never done or which I think can be done better. A few examples: How about a terminal program written in Action! that is designed to work well with CompuServe's conference mode? Or a GEM-like interface for DOS? Or a combined spreadsheet/database written in BASIC XE and commented liberally so that even beginners can see the methods used? I know I'll never do all of these, but they are challenges I'd *like* to tackle.

Rethinking The Problem

Moving to new languages on new machines is not always an advantage. For instance, in ST BASIC, strings cannot exceed 255 characters in length. Atari BASIC strings can be up to 32,767 characters long, if you have the memory available. (Yes, ST BASIC allows string arrays, but so do BASIC XL, BASIC XE, and Atari Microsoft BASIC.) There are many other examples.

Another factor is that the speed and power of the newer machines is of little advantage for some applications. Other than missing an 80-column screen, I can use CompuServe or various bulletin boards just as well with my $100 computer as I can with the company's $1,000 machine. Besides, the modem for the $100 computer is cheaper. And by the time you read this, Atari may have released its 80-column adapter for the eight-bit line.

Suppose you're writing a program that *does* need more speed, however. What can you do other than buy a newer, faster computer? Well, you could buy a better, faster language. That's a lot cheaper than buying a new computer—for which you still might need an extra language or two. On the other hand, maybe you don't have to buy anything at all; maybe you just need to rethink your solution to the problem. Let me show you what I mean.

Program 1 is very similar to one which I found in a recent user group newsletter. The author was responding to a member's inquiry about writing a routine to shuffle a deck of cards. As you know, when BASIC gives you a random number, there's no guarantee it won't give you that same random number twice, perhaps even several times. For a quick example, type the following line and press RETURN:

```
FOR I=0 TO 9:PRINT INT(10*RND
(0)):NEXT I
```

This asks for ten random numbers in the range 0 to 9. Did you actually get ten different numbers? The odds are very much against it.

The Super Shuffle

Program 1 demonstrates this problem by dealing out an entire deck of 52 cards. As each card is dealt (by suit S and rank R, line 210), its spot in the C (card) array is marked. Then, if the random number generator picks that card again, the pick is ignored (line 230). The only things I added to the original routine are the counters (C and T) which count how many picks it takes to get each card and the entire deck. Can you guess how many picks it takes to get the entire deck? In 50 tests, it took a minimum of 128 picks and a maximum of 457, with the average around 220. The

result, as you'll see if you run the program, is that it can take as long as five or six seconds to pick a card.

Now look at Program 2, which does exactly the same job but never takes more than one pick to get the next card in the deck. It works by using a single string (CARD$) to represent the entire deck. When it gets a random number from 1 to 52, the program removes the corresponding "card" from the "deck" (lines 400 and 410). The next time it picks a card, it gets a random number from 1 to 51. Each time the computer gets a card, the range of random numbers gets smaller. Simple. And it works by taking advantage of the string operations in Atari BASIC.

The point of this exercise is to show that sometimes the best way to fix a slow or inefficient program is to rethink it and then rewrite it. I'd be willing to bet that Program 2 on an eight-bit Atari runs faster than Program 1 on an Atari ST. If you have access to both machines, you might want to try it. And try improving your own programs. (Even while writing this column, I found a way to improve Program 2 even more. Can you find it?)

One last comment: Notice the readability of the two programs. Which one is cryptic and which one almost explains itself? Meaningful variable names can add a great deal of value to any program.

For instructions on entering these listings, please refer to "COMPUTE!'s Guide to Typing In Programs" in this issue of COMPUTE!.

Program 1: Slow Shuffle

```
HD 100 DIM C(4,13)
DP 110 DIM R$(13):R$="A23456
       789TJQK"
IG 120 DIM S$(4):S$="{;}{,}
       {.}{P}"
AH 130 FOR S=1 TO 4:FOR R=1
       TO 13:C(S,R)=0
PC 140 NEXT R:NEXT S
CE 200 FOR I=1 TO 52:C=0
HC 210 S=INT(RND(0)*4)+1:R=I
       NT(RND(0)*13)+1
LD 220 C=C+1
PF 230 IF C(S,R)<>0 THEN 210
BJ 240 C(S,R)=1
DK 250 T=T+C
LB 260 PRINT I;:POKE 85,15-L
       EN(STR$(C)):PRINT C;"
        PICK(S) TO GET ";R$(
       R,R);" OF ";S$(S,S)
CB 270 NEXT I
PJ 280 PRINT :PRINT "TOTAL P
       ICKS: ";T
```

Program 2: Fast Shuffle

```
ED 100 REM === SET UP VARIAB
       LES,ETC. ===
SC 110 DIM CARD$(52)
BA 120 DIM SUITS$(8*4)
CA 130 SUITS$="SPADES  HEART
       S  CLUBS{3 SPACES}DIA
       MONDS"
GB 140 DIM SUIT$(8)
PH 150 DIM RANKS$(4*5)
EE 160 RANKS$="ACE  JACK KIN
       G QUEEN"
EI 170 DIM RANK$(5)
GH 200 REM === SET UP THE DE
       CK ===
AM 210 FOR CARD=1 TO 52
DB 220 CARD$(CARD)=CHR$(CARD
       )
DD 230 NEXT CARD
IN 240 DECKSIZE=52
MP 300 REM === DEAL 52 CARDS
        ===
AN 310 FOR CARD=1 TO 52
HC 320 PICK=INT(DECKSIZE*RND
       (0))+1
LP 330 PICKED=ASC(CARD$(PICK
       ))-1
JI 340 SUIT=INT(PICKED/13)
LB 350 RANK=PICKED-13*SUIT
IF 360 SUIT$=SUITS$(SUIT*8+1
       ,SUIT*8+8)
HD 370 IF RANK<4 THEN RANK$=
       RANKS$(RANK*5+1,RANK*
       5+5)
LK 380 IF RANK>=4 THEN RANK$
       =STR$(RANK-2)
DK 390 PRINT "Picked: ";RANK
       $;" OF ";SUIT$
AM 400 IF PICK<DECKSIZE THEN
        CARD$(PICK)=CARD$(PI
       CK+1)
EA 410 IF PICK=DECKSIZE THEN
        CARD$(PICK)=""
NF 420 DECKSIZE=DECKSIZE-1
PA 430 NEXT CARD
```

©

Bill Wilkinson

Odd Facets Of GEM

This month we're going to explore a handful of quirks in the GEM desktop: things you can do to make your system more useful...things you can do to make your system crash! Since I'm a natural-born pessimist, let's start with the crash.

The bug I'm about to demonstrate infests the ROM (Read Only Memory) version of the TOS operating system. Even so, if you don't already have the TOS ROMs, get them today. The difference in overall system performance and capability is only a little short of great.

To see this bug, simply boot your system and bring up the Control Panel. Select either the date or time field. Then type an underline character (SHIFT-hyphen). Watch your system bomb. The only recovery is to press the reset button or turn off the power.

Problem: The Control Panel is a form of dialog box, and it uses what are known as *editable text fields* to display and let you modify the date and time. An editable text field is designed to restrict the user to typing certain characters. For example, the date and time fields of the Control Panel are editable fields which allow only numbers to be typed. Unfortunately, somehow a bug crept into the ROM-based TOS. Anytime you edit a numeric-only field, typing the underline causes something nasty to occur. Editable fields for filenames have a similar, though usually nonfatal, problem.

Solution: A GEM application program that needs to accept numeric-only input from the user has two choices: (1) Use an editable field which allows *any* character and then validate the user's input after the dialog box returns; (2) Retrieve keystrokes one at a time, checking them on the fly, and print only the valid ones on the screen. The former solution is kind of ugly because the user doesn't get immediate response to incorrect input. The latter solution is a lot of work. Take your pick.

Modifying DESKTOP.INF

Many of you already know how to customize the GEM desktop so your preferences appear automatically when you boot up the ST. When you select the *Save Desktop* item under the *Options* menu, GEM saves a file to the disk in drive A called DESKTOP.INF which stores these preferences. You can rearrange the icons on the screen, change screen colors, resize the windows, and so on, and GEM remembers it all for you.

DESKTOP.INF is an ordinary ASCII file, so it can be modified with most text editors and word processors. This lets you personalize GEM even more. (See "ST Hints & Tips," COMPUTE!, June 1986.)

The first thing we'll do is the easiest. Using a text editor or word processor that handles ASCII files, load and examine DESKTOP.INF. You should see one or two lines which contain the words FLOPPY DISK (among other things). These are the labels which appear beneath the disk icons. I usually rename the labels *—Top—Disk—* and *Bottom—Disk.* (I've used dashes here to show where I typed a space—magazine typesetting sometimes makes it hard to indicate spaces.)

Save the modified file back on disk in ASCII format. The next time you boot from that disk, the names should appear as you have modified them. Just for fun, sometimes I change the name of the trash can to *Junk!* or *Garbage* or something equally silly.

Rearranging Files

There are even more interesting things you can do with DESKTOP.INF. If, like me, you have a disk or subdirectory in which you do most of your work, you'll soon find that you can't see all of the filenames or icons on the screen at once. Although it's a minor nuisance, it always seems that the files (or, more likely, programs) which I want the most are always off the screen. How can we force them back on the screen? Preferably in the upper-left position?

One solution, since the default display mode under the *Show* menu is *Sort by Name*, is to name your favorite files AARDVARK.PRG or AAABASIC.PRG. But that's kind of messy. A better method might be to choose *Sort by Date* if you could change the file's creation date. But I think Mark Rose (of Optimized Systems Software) has hit upon the best scheme.

First, he chooses *Sort by Type.* Second, he renames his most-used programs so they have no type (filename extension) at all. Third, he loads DESKTOP.INF and adds a line or so. To figure out exactly what to add, look for a line in DESKTOP.INF similar to this:

```
#G 03 FF *.PRG@ @
```

This line tells the desktop that all files which match the *.PRG specifier are GEM (G) program files. Now, let's say the program you want to appear at the top-left of the screen was called PASCAL.PRG and has been renamed to simply PASCAL. You would add this line to the end of the DESKTOP.INF file:

```
#G 03 FF PASCAL.@ @
```

This tells GEM that PASCAL is actually a GEM-based program. Neat, huh? What's more, you can do this for several files. However, I *do not* recommend using the *. wildcard in such a line—general untyped files end up looking like programs, a dangerous practice. ©

A Beginning Reading Program

When my youngest son was learning to read, I wrote lists of words for him to practice with. The word lists used a certain word ending coupled with various beginning letters—such as AT, BAT, CAT, FAT, HAT, MAT, PAT, RAT, SAT, TAT, VAT, and THAT. After I wrote the same list several times, I realized this was another idea for a computer application.

Not only can the TI-99/4A print the words nicely on the screen, but with the TI Speech Synthesizer it can also speak the words aloud. This month's example program, "Reading Practice," takes advantage of this feature. Therefore, it requires both the TI Speech Synthesizer and the Terminal Emulator 2 command module. Insert the module and select 1 for TI BASIC to gain access to speech.

Reading Practice

Notice line 154 in the program listing below. This OPENs the channel for speech, so any subsequent PRINT #1 statement will speak the word. The words I selected for examples are all pronounced correctly by the speech synthesizer as they are spelled. If you add words, check their pronunciation; you may need to add a routine so the word is pronounced correctly.

If you want to run this program without speech, omit all statements that refer to file 1 or insert a REM in front of each of these commands. Another alternative is to make speech an option; insert IF-THEN statements before the speech commands.

Due to space limitations, I had to keep Reading Practice rather short. Feel free to add graphics, sound, and additional word lists. I included just a few for examples. You could also modify the program to offer the student various groupings of words.

When you run Reading Practice, you'll see a word ending in lowercase letters. To hear the word pronounced aloud, press the space bar. To go to the next word, press ENTER. The screen clears and you'll see a word with a beginning letter and that word ending.

The words are in the DATA statements starting at line 198. First there is a word ending, then all the beginning letters that go with the word ending and create real words. ZZZ indicates no more beginning letters for that ending. At the end of all the lists, @@@ indicates the end of data.

Lines 126–146 redefine characters to draw larger lowercase letters. Be careful typing the DATA statements; they contain the character definitions. If you have my program for lowercase letters (COMPUTE!, August 1983), you can take a shortcut. First, load that program and delete all the PRINT statements. Type RES 126,2 to renumber the lines. Then add the rest of the Reading Practice program.

Lines 20–110 contain subroutines to draw the 26 letters of the alphabet. The variable R is the main row number and C is the column number. If a lowercase letter has an ascender or descender, more than one character is required to draw the letter. Line 174 goes to the proper subroutine for each letter in the word. Since there are 26 possible letters, I needed low line numbers to use a single ON-GOSUB statement (otherwise more complex logic and several ON-GOSUB statements would be required). Therefore, the subroutines are near the beginning of the program, and the program starts with line 2 and increments by 2. To type in this program using the automatic numbering feature, use NUM 2,2.

TI BASIC String-Handling

Reading Practice uses several string functions. It reads the letters from the DATA statements into X$ for the word ending and B$ for the beginning letters. W$ holds the word to be printed and read. LEN(W$) returns the length of the word (or word ending). Lines 170–178 contain a FOR-NEXT loop that executes for each letter of the word. SEG$ gets one character at a time from the word, and ASC returns the ASCII value of that letter. In TI BASIC, the ampersand symbol (&) is used to combine strings. (See "The Beginner's Page," June 1986.)

Line 194 combines the beginning letter B$ with the word ending X$ to form the word W$. Line 186 adds a period to the word to change its inflection for the speech synthesizer. You may prefer to simply use W$.

Versions of BASIC on other computers generally use the string functions RIGHT$, LEFT$, and MID$ for extracting sections of strings. The equivalent in TI BASIC is SEG$. SEG$(W$,A,B) looks at the string W$, starts with character number A, and returns B number of characters. For example, let's assume that W$="RICHARD". The Microsoft BASIC statement LEFT$(W$,4) is translated into TI BASIC as SEG$(W$,1,4), which yields RICH. The statement RIGHT$(W$,4) is translated as SEG$(W$,LEN(W$)−4+1,4), which yields HARD. And the statement MID$(W$,3,2) is translated as SEG$(W$,3,2), which yields CH. (See "The Beginner's Page," April 1986.)

In the Reading Practice program, SEG$(W$,P,1) is in a loop where P starts at 1 and goes to L, which is the number of characters in the word. This function, then, gets one letter at a time, in order, from the word. The variable A is the ASCII value of that character minus 64 to yield numbers from 1

to 26 for the relative letter. Line 174 then uses ON-GOSUB to go to the proper subroutine to draw the corresponding lowercase letter.

If you want to save typing effort, you may get a copy of this program by sending a blank cassette or disk, a stamped, self-addressed mailer, and $3 to:

C. Regena
P.O. Box 1502
Cedar City, UT 84720

Be sure to specify the title "Reading Practice." This program is available for the TI computer only.

Reading Practice

```
2 REM   READING
4 REM   REQUIRES TERMINAL
6 REM     EMULATOR 2
8 REM   REQUIRES SPEECH
10 REM     SYNTHESIZER
12 REM   ** WITHOUT SPEECH--

14 REM    REMOVE STATEMENTS
16 REM    CONTAINING "#1"
18 GOTO 112
20 CALL HCHAR(R,C,A+96)
22 RETURN
24 CALL HCHAR(R-1,C,104)
26 GOTO 20
28 CALL HCHAR(R-1,C,100)
30 CALL HCHAR(R,C,97)
32 RETURN
34 CALL HCHAR(R-1,C,102)
36 CALL HCHAR(R,C,108)
38 RETURN
40 CALL HCHAR(R,C,97)
42 CALL HCHAR(R+1,C,103)
44 RETURN
46 CALL HCHAR(R-1,C,104)
48 CALL HCHAR(R,C,110)
50 RETURN
52 CALL HCHAR(R-1,C,105)
54 CALL HCHAR(R,C,108)
56 RETURN
58 GOSUB 52
60 CALL HCHAR(R+1,C,106)
62 RETURN
64 CALL HCHAR(R-1,C,104)
66 GOTO 20
68 CALL VCHAR(R-1,C,108,2)
70 RETURN
72 CALL HCHAR(R,C,110)
74 C=C+1
76 CALL HCHAR(R,C,109)
78 RETURN
80 CALL HCHAR(R,C,98)
82 CALL HCHAR(R+1,C,112)
84 RETURN
86 CALL HCHAR(R,C,97)
88 CALL HCHAR(R+1,C,113)
90 RETURN
92 CALL HCHAR(R-1,C,116)
94 CALL HCHAR(R,C,108)
96 RETURN
98 CALL HCHAR(R,C,118)
100 C=C+1
102 CALL HCHAR(R,C,119)
104 RETURN
106 CALL HCHAR(R,C,118)
108 CALL HCHAR(R+1.C.121)
110 RETURN
112 CALL CLEAR
114 PRINT "** READING PRACT
    ICE **"
116 PRINT ::::"READ THE WOR
    D ON THE SCREEN."
118 PRINT ::"PRESS THE SPAC
    E BAR TO HEAR"
120 PRINT ::"THE WORD."
122 PRINT ::"PRESS <ENTER>
    TO GO TO THE"
124 PRINT ::"NEXT WORD."::::
126 FOR C=97 TO 122
128 READ C$
130 CALL CHAR(C,C$)
132 NEXT C
134 REM  REDEFINE LOWERCASE

136 DATA 3D4381818181433D,8
    CC281818181C28C,3C42808
    08080423C,0000010101010
    101,3C4281FF8080423C
138 DATA 0609080808083E,0
    101010141221C,000080808
    080808,00000008,0808080
    808887,8890A0C0A0908884
140 DATA 0808080808080808,7
    884020202020202,8CC2818
    181818181,3C42818181814
    23C,80808080808,0101010
    10101
142 DATA 8CC281808080808,3C
    42403C0202423C,00000808
    08087F08,81818181818143
    3D,4141222214140808,040
    488885050202
144 DATA 8244281028448282,1
    0102020404,7F0204081020
    407F
146 REM
148 PRINT "NOW PRESS <ENTER
    > TO START."
150 CALL KEY(0,K,S)
152 IF K<>13 THEN 150
154 OPEN #1:"SPEECH",OUTPUT
156 R=10
158 READ X$
160 IF X$="@@@" THEN 216
162 W$=X$
164 CALL CLEAR
166 L=LEN(W$)
168 C=12
170 FOR P=1 TO L
172 A=ASC(SEG$(W$,P,1))-64
174 ON A GOSUB 20,24,20,28,
    20,34,40,46,52,58,64,68
    ,72,20,20,80,86,20,20,9
    2,20,20,98,20,106,20
176 C=C+2
178 NEXT P
180 CALL KEY(0,K,S)
182 IF K=13 THEN 190
184 IF K<>32 THEN 180
186 PRINT #1:W$&"."
188 GOTO 180
190 READ B$
192 IF B$="ZZZ" THEN 158
194 W$=B$&X$
196 GOTO 164
198 DATA AT,B,C,F,H,M,P,R,S
    ,T,V,TH,ZZZ
200 DATA AN,B,C,F,M,P,R,T,V
    ,TH,ZZZ
202 DATA ED,B,F,L,R,W,ZZZ
204 DATA IN,B,F,K,P,S,T,W,Z
    ZZ
206 DATA IT,B,F,H,K,L,P,S,W
    ,ZZZ
208 DATA OG,C,D,F,H,J,L,ZZZ
210 DATA UG,B,D,H,J,L,M,R,T
    ,ZZZ
212 DATA AND,B,H,L,S,ZZZ
214 DATA @@@
216 CALL CLEAR
218 CLOSE #1
220 END
```

©

Programming In Modula-2

There are a plethora of programming languages for the Amiga, giving programmers and developers a wide choice of programming styles and systems. There are two versions of BASIC (MetaComCo's ABasiC and Microsoft's Amiga BASIC), two C compilers (Manx Aztec C and Lattice C), a macro assembler/editor, two versions of Pascal, and even an implementation of LISP. Numerous programming tools, such as editors and debuggers, are also available.

A relative newcomer to the scene, TDI Modula-2, is now getting some attention. Some programmers consider it easier to learn and use than C—since it shares many of the high-level aspects of Pascal—while still retaining a machine-level interface for maximum efficiency.

Modula-2 is a descendent of the language Modula, which in turn is a descendent of Pascal. Nicklaus Wirth, the inventor of Pascal, designed Modula from the roots of Pascal, but purposely kept it very simple so that it could be used with very small computers—primarily for controlling hardware devices such as robot arms. The original Modula had little application outside a very specialized world, so Wirth put back most of the features of Pascal to create Modula-2. TDI has worked directly with Wirth to implement versions of Modula-2 for the Amiga and Atari ST.

Software Chips

The concept of Modula-2 is echoed in its name. It is a language designed specifically for the techniques of modular programming, just as Pascal was designed to make structured programming convenient and elegant. Modular programming—the art of breaking a large, complex problem into small, independent tasks—is at the heart of all programming, but Modula-2 tries to bring to software the modularity inherent in computer hardware, based on off-the-shelf chips and components. With "software chips," Wirth envisioned, software technology could advance apace with the remarkable speed of hardware evolution.

If software chips are possible, they have to be based on program modules that can be truly independent, hence, individually testable. You can compile a module without having to recompile the entire program. A module, once developed, becomes a "black box" routine that accepts input and/or provides output. You no longer need to know how this module works internally to use it—you just plug it in and go. Writing a program becomes a task of putting together these building blocks in the right way without ever needing to reinvent the wheel. Why solve a problem when someone else has already found the solution?

Modula-2 comes with a standard library containing modules for input/output, math routines, and access to special machine features. You use only the routines you're interested in, and only these routines (and the underlying routines they are based on) need to be included in your compiled code. This lets you control the size of your final program.

You can easily add your own library modules. First, you write the *definition module*, which simply contains the procedure headers that specify the inputs and outputs of a module. The definition module primarily specifies the names of these procedures. It compiles to a symbol file for use by the compiler. The *implementation module* contains the actual code of the module. You compile the definition module separately from the implementation module.

You can change and recompile the implementation module without changing the definition module, as long as your procedure headings remain the same. When you're referencing library modules, the compiler can check the compact, compiled symbol file rather than the full-length definition module, speeding up compilation. After compilation, a linker combines your main program with the compiled implementation modules to create the final executable program.

Reminiscent Of Pascal

One of the best ways to learn about a language is to study an example program. The program accompanying this column is written to demonstrate some of the features of Modula-2 without getting bogged down in tricky algorithms. It's a simple guess-my-number game. The RandomNumbers module thinks of a number from 1 to 100. The program then gives you ten tries to guess the number, helping out with hints. If you guess too high, the program recommends that you try a smaller number. If you guess too low, you should try a higher number.

Here's how the program works. The first line declares the name of the module. Next, the IMPORT statements specify which external library calls we'll be using. Then we declare the variables. We define the procedure SkipEOL, used to strip away the rest of a line after getting a single-character response. The main loop follows, enclosed by the keywords BEGIN and END. (All Modula-2 keywords must be typed in uppercase, which can be annoying.)

Most of the program looks very much like Pascal, especially the use

of := for assignments and the required semicolon at the end of each logical line. Also, you won't find GOTO anywhere in this or any Modula-2 program. Instead, you can control looping and program execution with statements like LOOP-EXIT-END, WHILE-END, REPEAT-UNTIL, and IF-THEN-ELSE-END.

You might be interested to know that this program compiles in 35 seconds when the source code is stored in the RAM disk; it takes 37 seconds to compile when the source code is stored on a floppy disk. Linking takes 45 seconds from the RAM disk, and just one minute from a floppy disk. This is quite a bit faster than Lattice C and compares well with Aztec C.

There's much more to Modula-2 than this discussion can encompass. The language even permits procedures to run as multitasking programs. Our example doesn't show how easily Modula-2 can take advantage of the Amiga operating system—even a small program would be too large to demonstrate here—but the interface is similar to C's, using Pascal-style RECORDs instead of C structures. It's possible to develop modules that support the Amiga operating system on a higher level, using calls like Screen(320,200,5) to open a custom screen as opposed to filling in the blanks of a NewScreen structure, opening the Intuition library, and calling OpenScreen( ). Some high-level modules are included in the library. When these modules are developed and shared between Modula-2 programmers, Amiga programming in Modula-2 can seem almost as easy as in BASIC, but with every advantage of a modern compiled language.


```
MODULE Example;

FROM InOut IMPORT EOL,Read,ReadInt,ReadString,
     WriteLn,WriteString,WriteInt;
FROM RandomNumbers IMPORT Random;

VAR
 MyNum, Guess, Tries : INTEGER;
 Again : CHAR;

(* Skips until end of line is reached *)
PROCEDURE SkipEOL;
  VAR temp : CHAR;
BEGIN
  REPEAT
    Read(temp);
  UNTIL temp=EOL;
END SkipEOL;

(* The main loop *)
BEGIN
  LOOP
    MyNum := Random(100)+1;
    WriteString("I'm thinking of a ");
    WriteString("number from 1 to 100");
    WriteLn;
    Tries :=0;
    LOOP
      Tries := Tries+1;
      IF Tries>10 THEN EXIT; END;
      WriteString("Guess #");
      WriteInt(Tries,2);
      WriteString("? ");
      ReadInt(Guess);
      WriteLn;
      IF (Guess=MyNum) OR (Guess=0) THEN EXIT; END;
      IF Guess<MyNum THEN
        WriteString("Try a larger number.");
      ELSE
        WriteString("Try a smaller number.");
      END; (* IF *)
      WriteLn; WriteLn;
    END; (* LOOP *)
    IF Tries>10 THEN
      WriteString("You only get 10 tries!");
    END; (* IF *)
    IF Guess=MyNum THEN
      WriteString("You guessed my number!");
      WriteLn;
      WriteString("After ");
      WriteInt(Tries,2);
      WriteString(" tries.");
    END; (* IF *)
    WriteLn; WriteLn;
    WriteString("Play again? (Y/N): ");
    Read(Again);
    SkipEOL; (* skip ahead to next line *)
    IF (Again='N') OR (Again='n') THEN EXIT; END;
  END; (* LOOP *)
END Example.
```

©

Donald B. Trivette

Hard Disks And The Home PC

Technology marches on: The price of a hard disk drive—a $2,000 luxury just a few years ago—has fallen to a point where it's becoming affordable for home-based computers. Mail-order houses are offering internal hard disks for less than $400, and prices of hard disk cards are coming down as well. If you're thinking about upgrading your IBM PC with one of these super storage devices, here are some points to consider.

Hard disk, fixed disk, and *Winchester disk* are all names for the same thing—a device with a rigid magnetic disk permanently sealed in a box. Because it's sealed from airborne contaminants and has a hard rather than a flexible surface, it can record larger amounts of data than a floppy disk, and the data can be read or written much faster. The smallest hard disks commonly in use store 10 megabytes of data—that's 10,240 kilobytes, or 10,485,760 characters. In comparison, a standard IBM floppy disk stores only 360 kilobytes—368,640 characters. Hard disks are available for the PC with capacities of 20, 50, and even 100 megabytes, although a home user isn't likely to need more than 10 megabytes.

Hard disks come in two forms: the internal type that fits into one of the spaces once occupied by a floppy disk drive (half-height or full-height), and a newer configuration called a hard disk card that squeezes the disk onto a printed circuit board which plugs into one of the PC's expansion slots.

Can You Spare A Slot?

Every hard disk must have a hard disk controller. An ordinary internal hard disk requires a separate controller board connected via cables to the drive. One advantage of the hard disk cards is that both the disk and the controller are a single compact unit. Either way, one expansion slot is used. But there is an alternative for internal models. For about $50 extra, you can order a controller that runs existing floppy disk drives as well as the hard drive. This allows you to remove (and scrap) your floppy disk controller board and free up one expansion slot. None of the hard disk cards introduced so far can control floppy disk drives.

Another factor to consider is power consumption. Hard disk cards are designed to work with the 63-watt power supply found in the IBM PC; many of the internal hard disks were designed for the PC-XT, which has a larger power supply. Frankly, if your PC is already brimming with boards—parallel, serial, color/graphics, and game ports as well as memory expansion—you should strongly consider replacing the old power supply no matter which type of hard disk you install. The operation is as simple and safe as removing some screws and unplugging some wires. A new 135-watt power supply can be purchased for as low as $90.

Of course, speed and reliability are prime considerations when investing in a hard disk. Although reliability is difficult to measure without an industry-wide mean-time-between-failure test, there are some things you can check. An oxide coating on the disk surface is more stable and thus more reliable than plated or sputtered media. The type of actuator that moves the read/write heads across the surface of the platter not only affects speed, but also influences accuracy. A voice-coil actuator is faster and more reliable (and more expensive) than the more common stepper-motor actuator. Therefore, look for a hard disk with an oxide coating and voice-coil actuator.

Fast, Faster, Fastest

Speed varies greatly depending on the make and model. Consider the results of a test which measures a mixture of 1,000 sequential and random accesses—a test that is typical of how real computer programs use a hard disk. The times range from 12 milliseconds for a high-performance internal drive like Core International's AT line, to more then 100 milliseconds for some of the inexpensive hard disk cards. A hard disk for home use—where you don't need top performance—should have a sequential/random access time in the 30- to 60-millisecond range.

Cost is usually a major consideration when selecting a hard disk for home use. Hard disk cards cost as little as $550 for a 10-megabyte unit to as much as $1,200 for 20 megabytes. Internal 20-megabyte drives, including the controller board, are generally in the $400–$600 range; 10-megabyte models cost about $100 less. The best advice here is not to choose a disk by cost alone. Consider all the factors.

My own PC has a 20-megabyte internal Seagate drive, a new 135-watt power supply, and a controller that also runs two half-height floppy drives. It cost about $650 and took two hours to install. An alternative I'd be comfortable with is Plus Development's 10-megabyte Hardcard; it's designed for those who want a quick, simple installation and who don't want to fool with a new power supply. The Hardcard, at about $900, is a well-engineered solution to upgrading your computer's mass storage capabilities.

One final caveat: Whether you spend ten minutes or two hours installing the hardware, plan to spend lots more time learning to use the DOS commands that are necessary to manage files on a hard disk. ©

MLX Machine Language Entry Program For Commodore 64

Ottis Cowper, Technical Editor

"MLX" is a labor-saving utility that allows will help you enter machine language program listings without error. MLX is required to enter all Commodore 64 machine language programs published in COMPUTE!.

Type in and save some copies of MLX (you'll want to use it to enter future ML programs from COMPUTE!, COMPUTE!'s GAZETTE, and COMPUTE! books). When you're ready to enter an ML program, load and run MLX. You'll be asked for a starting address and an ending address. These addresses should appear in the article accompanying the MLX-format program listing you're typing.

If you're unfamiliar with machine language, the addresses (and all other values you enter in MLX) may appear strange. Instead of the usual decimal numbers you're accustomed to, these numbers are in *hexadecimal*—a base 16 numbering system commonly used by ML programmers. Hexadecimal—hex for short—includes the numerals 0–9 and the letters A–F. But don't worry—even if you know nothing about ML or hex, you should have no trouble using MLX.

After you enter the starting and ending addresses, you'll be offered the option of clearing the workspace. The data you enter with MLX is kept in a special reserved area of memory; clearing this workspace fills the reserved area with zeros, which makes it easier to find where you left off typing if you enter the listing in several sessions. Choose this option if you're starting to enter a new listing. If you're continuing a listing that's partially typed from a previous session, there's no point in clearing the workspace, since the data you load in will fill the area with whatever values were in workspace memory at the time of the last Save.

At this point, functions menu will appear. If you're just starting to type in a program, pick the first option, ENTER DATA, by pressing the E key. You'll be asked for an address; type the four-digit number at the start of the first line of the program listing. If you've already typed in part of a program, be sure to load the partially completed program before you resume entry, then choose the ENTER DATA option and type the line number where you left off typing at the end of the previous session. In any case, make sure the address you enter corresponds to the address of a line in the listing. Otherwise, you'll be unable to enter the data correctly. If you pressed E by mistake, you can return to the command menu by pressing RETURN alone when asked for the address. (You can get back to the menu from most options by pressing RETURN with no other input.)

Entering A Listing

Once you're in Enter mode, MLX prints the address for each program line for you. You then type in all nine numbers on that line, beginning with the first two-digit number after the colon (:). Each line represents eight data bytes and a checksum. Although an MLX-format listing appears similar to the "hex dump" listings from a machine language monitor program, the extra checksum number on the end allows MLX to check your typing.

When you enter a line, MLX recalculates the checksum from the eight bytes and the address and compares this value to the number from the ninth column. If the values match, you'll hear a bell tone, the data will be added to the workspace area, and the prompt for the next line of data will appear. But if MLX detects a typing error, you'll hear a low buzz and see an error message. The line will then be redisplayed for editing.

Invalid Characters Banned

Only a few keys are active while you're entering data, so you may have to unlearn some habits. You *do not* type spaces between the columns; MLX automatically inserts these for you. You *do not* press RETURN after typing the last number in a line; MLX automatically enters and checks the line after you type the last digit.

Only the numerals 0–9 and the letters A–F can be typed in. If you press any other key (with some exceptions noted below), you'll hear a warning buzz. MLX checks for transposed characters. If you're supposed to type in A0 and instead enter 0A, MLX will catch your mistake. There is one error that can slip past MLX: Because of the checksum formula used, MLX won't notice if you accidentally type FF in place of 00, and vice versa. And there's a very slim chance that you could garble a line and still end up with a combination of characters that adds up to the proper checksum. However, these mistakes should not occur if you take reasonable care while entering data.

Editing Features

To correct typing mistakes before finishing a line, use the INST/DEL key to delete the character to the left of the cursor. (The cursor-left key also deletes.) If you mess up a line really badly, press CLR/HOME to start the line over. The RETURN key is also active, but only before any data is typed on a line. Pressing RETURN at this point returns you to the command menu. After you type a character of data, MLX disables RETURN until the cursor returns to the start of a line. Remember, you can press CLR/HOME to quickly get to a line number prompt.

More editing features are available when correcting lines in which MLX has detected an error. To make corrections in a line that MLX has redisplayed for editing, compare the line on the screen with the one printed in the listing, then move the cursor to the mistake and type the correct key. The cursor left and right keys provide the normal cursor controls. (The INST/DEL key now works as an alternative cursor-left key.) You cannot move left beyond the first character in the line. If you try to move beyond the rightmost character, you'll reenter the line. During editing, RETURN is active; pressing it tells MLX to recheck the line. You can press the CLR/HOME key to clear the entire line if you want to start from scratch, or if you want to get to a line number prompt to use RETURN to get back to the menu.

Display Data

The second menu choice, DISPLAY DATA, examines memory and shows the contents in the same format as the program listing (including the checksum). When you press D, MLX asks you for a starting address. Be sure that the starting address you give corresponds to a line number in the listing. Otherwise, the checksum display will be meaningless. MLX displays program lines until it reaches the end of the program, at which point the menu is redisplayed. You can pause the display by pressing the space bar. (MLX finishes printing the current line before halting.) Press space again to restart the display. To break out of the display and

get back to the menu before the ending address is reached, press RETURN.

Other Menu Options

Two more menu selections let you save programs and load them back into the computer. These are SAVE FILE and LOAD FILE; their operation is quite straightforward. When you press S or L, MLX asks you for the filename. You'll then be asked to press either D or T to select disk or tape.

You'll notice the disk drive starting and stopping several times during a load or save. Don't panic; this is normal behavior. MLX opens and reads from or writes to the file instead of using the usual LOAD and SAVE commands. Disk users should also note that the drive prefix 0: is automatically added to the filename (line 750), so this should *not* be included when entering the name. This also precludes the use of @ for Save-with-Replace, so remember to give each version you save a different name.

Remember that MLX saves the entire workspace area from the starting address to the ending address, so the save or load may take longer than you might expect if you've entered only a small amount of data from a long listing. When saving a partially completed listing, make sure to note the address where you stopped typing so you'll know where to resume entry when you reload.

MLX reports the standard disk or tape error messages if any problems are detected during the save or load. (Tape users should bear in mind that Commodore computers are never able to detect errors during a save to tape.) MLX also has three special load error messages: INCORRECT STARTING ADDRESS, which means the file you're trying to load does not have the starting address you specified when you ran MLX; LOAD ENDED AT *address*, which means the file you're trying to load ends before the ending address you specified when you started MLX; and TRUNCATED AT ENDING ADDRESS, which means the file you're trying to load extends beyond the ending address you specified when you started MLX. If you see one of these messages and feel certain that you've loaded the right file, exit and rerun MLX, being careful to enter the correct starting and ending addresses.

The QUIT menu option has the obvious effect—it stops MLX and enters BASIC. The RUN/STOP key is disabled, so the Q option lets you exit the program without turning off the computer. (Of course, RUN/STOP-RESTORE also gets you out.) You'll be asked for verification; press Y to exit to BASIC, or any other key to return to the menu. After quitting, you can type RUN again and reenter MLX without losing your data, as long as you don't use the clear workspace option.

The Finished Product

When you've finished typing all the data for an ML program and saved your work, you're ready to see the results. The instructions for loading and using the finished product vary from program to program. Some ML programs are designed to be loaded and run like BASIC programs, so all you need to type is LOAD *"filename"*,8 for disk or LOAD *"filename"* for tape, and then RUN. Such programs will usually have a starting address of 0801. Other programs must be reloaded to specific addresses with a command such as LOAD *"filename"*,8,1 for disk or LOAD *"filename"*,1,1 for tape, then started with a SYS to a particular memory address. The most common starting address for such programs is 49152, which corresponds to MLX address C000. In either case, you should always refer to the article which accompanies the ML listing for information on loading and running the program.

An Ounce Of Prevention

By the time you finish typing in the data for a long ML program, you may have several hours invested in the project. Don't take chances—use our "Automatic Proofreader" to type MLX, and then test your copy *thoroughly* before first using it to enter any significant amount of data. Make sure all the menu options work as they should. Enter fragments of the program starting at several different addresses, then use the Display option to verify that the data has been entered correctly. And be sure to test the Save and Load options several times to ensure that you can recall your work from disk or tape. Don't let a simple typing error in MLX cost you several nights of hard work.

MLX

For instructions on entering this listing, please refer to "COMPUTE!'s Guide to Typing In Programs" in this issue of COMPUTE!

```
EK 100 POKE 56,50:CLR:DIM IN$,
       I,J,A,B,A$,B$,A(7),N$
DM 110 C4=48:C6=16:C7=7:Z2=2:Z
       4=254:Z5=255:Z6=256:Z7=
       127
CJ 120 FA=PEEK(45)+Z6*PEEK(46)
       :BS=PEEK(55)+Z6*PEEK(56
       ):H$="0123456789ABCDEF"
SB 130 R$=CHR$(13):L$="{LEFT}"
       :S$=" ":D$=CHR$(20):Z$=
       CHR$(0):T$="{13 RIGHT}"
CQ 140 SD=54272:FOR I=SD TO SD
       +23:POKE I,0:NEXT:POKE
       {SPACE}SD+24,15:POKE 78
       8,52
FC 150 PRINT"{CLR}"CHR$(142)CH
       R$(8):POKE 53280,15:POK
       E 53281,15
EJ 160 PRINT T$" {RED}{RVS}
       {2 SPACES}{8 @}
       {2 SPACES}"SPC(28)"
       {2 SPACES}{OFF}{BLU} ML
       X II {RED}{RVS}
       {2 SPACES}"SPC(28)"
       {12 SPACES}{BLU}"
FR 170 PRINT"{3 DOWN}
       {3 SPACES}COMPUTE!'S MA
       CHINE LANGUAGE EDITOR
       {3 DOWN}"
JB 180 PRINT"{BLK}STARTING ADD
       RESS{4}";:GOSUB300:SA=A
       D:GOSUB1040:IF F THEN1B
       0
GF 190 PRINT"{BLK}{2 SPACES}EN
       DING ADDRESS{4}";:GOSUB
       300:EA=AD:GOSUB1030:IF
       {SPACE}F THEN190
KR 200 INPUT"{3 DOWN}{BLK}CLEA
       R WORKSPACE [Y/N]{4}";A
       $:IF LEFT$(A$,1)<>"Y"TH
       EN220
PG 210 PRINT"{2 DOWN}{BLU}WORK
       ING...";:FORI=BS TO BS+
       EA-SA+7:POKE I,0:NEXT:P
       RINT"DONE"
DR 220 PRINTTAB(10)"{2 DOWN}
       {BLK}{RVS} MLX COMMAND
       {SPACE}MENU {DOWN}{4}":
       PRINT T$"{RVS}E{OFF}NTE
       R DATA"
BD 230 PRINT T$"{RVS}D{OFF}ISP
       LAY DATA":PRINT T$"
       {RVS}L{OFF}OAD FILE"
JS 240 PRINT T$"{RVS}S{OFF}AVE
        FILE":PRINT T$"{RVS}Q
       {OFF}UIT{2 DOWN}{BLK}"
JH 250 GET A$:IF A$=N$ THEN250
HK 260 A=0:FOR I=1 TO 5:IF A$=
       MID$("EDLSQ",I,1)THEN A
       =I:I=5
FD 270 NEXT:ON A GOTO420,610,6
       90,700,280:GOSUB1060:GO
       TO250
EJ 280 PRINT"{RVS} QUIT ":INPU
       T"{DOWN}{4}ARE YOU SURE
        [Y/N]";A$:IF LEFT$(A$,
       1)<>"Y"THEN220
EM 290 POKE SD+24,0:END
JX 300 IN$=N$:AD=0:INPUTIN$:IF
       LEN(IN$)<>4THENRETURN
KF 310 B$=IN$:GOSUB320:AD=A:B$
       =MID$(IN$,3):GOSUB320:A
       D=AD*256+A:RETURN
PP 320 A=0:FOR J=1 TO 2:A$=MID
       $(B$,J,1):B=ASC(A$)-C4+
       (A$>"@")*C7:A=A*C6+B
JA 330 IF B<0 OR B>15 THEN AD=
       0:A=-1:J=2
GX 340 NEXT:RETURN
CH 350 B=INT(A/C6):PRINT MID$(
       H$,B+1,1);:B=A-B*C6:PRI
       NT MID$(H$,B+1,1);:RETU
       RN
RR 360 A=INT(AD/Z6):GOSUB350:A
       =AD-A*Z6:GOSUB350:PRINT
       ":";
BE 370 CK=INT(AD/Z6):CK=AD-Z4*
       CK+Z5*(CK>Z7):GOTO390
PX 380 CK=CK*Z2+Z5*(CK>Z7)+A
JC 390 CK=CK+Z5*(CK>Z5):RETURN
QS 400 PRINT"{DOWN}STARTING AT
       {4}";:GOSUB300:IF IN$<>
       N$ THEN GOSUB1030:IF F
       {SPACE}THEN400
EX 410 RETURN
HD 420 PRINT"{RVS} ENTER DATA
       {SPACE}":GOSUB400:IF IN
       $=N$ THEN220
```

```
JK 430 OPEN3,3:PRINT
SK 440 POKE198,0:GOSU8360:IF F
        THEN PRINT IN$:PRINT"
       {UP}{5 RIGHT}";
GC 450 FOR I=0 TO 24 STEP 3:B$
       =S$:FOR J=1 TO 2:IF F T
       HEN B$=MID$(IN$,I+J,1)
HA 460 PRINT"{RVS}"8$L$;:IF I<
       24THEN PRINT"{OFF}";
HD 470 GET A$:IF A$=N$ THEN470
FK 480 IF(A$>"/"ANDA$<":")OR(A
       $>"@"ANDA$<"G")THEN540
MP 490 IF A$=R$ AND((I=0)AND(J
       =1)OR P)THEN PRINT 8$;:
       J=2:NEXT:I=24:GOTO550
KC 500 IF A$="{HOME}" THEN PRI
       NT 8$:J=2:NEXT:I=24:NEX
       T:F=0:GOTO440
MX 510 IF(A$="{RIGHT}")ANDF TH
       ENPRINT 8$L$;:GOTO540
GK 520 IP A$<>L$ AND A$<>D$ OR
       ((I=0)AND(J=1))THEN GOS
       U81060:GOTO470
HG 530 A$=L$+S$+L$:PRINT B$L$;
       :J=2-J:IF J THEN PRINT
       {SPACE}L$;:I=I-3
QS 540 PRINT A$;:NEXT J:PRINT
       {SPACE}S$;
PM 550 NEXT I:PRINT:PRINT"{UP}
       {5 RIGHT}";:INPUT#3,IN$
       :IF IN$=N$ THEN CLOSE3:
       GOTO220
QC 560 FOR I=1 TO 25 STEP3:B$=
       MID$(IN$,I):GOSUB320:IF
        I<25 THEN GOSUB380:A(I
       /3)=A
PK 570 NEXT:IF A<>CK THEN GOSU
       81060:PRINT"{8LK}{RVS}
       {SPACE}ERROR: REENTER L
       INE {4}":P=1:GOTO440
HJ 580 GOSUB1080:8=8S+AD-SA:FO
       R I=0 TO 7:POKE 8+I,A(I
       ):NEXT
QQ 590 AD=AD+8:IF AD>EA THEN C
       LOSE3:PRINT"{DOWN}{BLU}
       ** END OF ENTRY **{8LK}
       {2 DOWN}":GOTO700
GQ 600 F=0:GOTO440
QA 610 PRINT"{CLR}{DOWN}{RVS}
       {SPACE}DISPLAY DATA ":G
       OSUB400:IP IN$=N$ THEN2
       20
RJ 620 PRINT"{DOWN}{8LU}PRESS:
        {RVS}SPACE{OFP} TO PAU
       SE, {RVS}RETURN{OFF} TO
        8REAK{4}{DOWN}"
KS 630 GOSU8360:B=8S+AD-SA:FOR
       I=8TO B+7:A=PEEK(I):GOS
       U8350:GOSU8380:PRINT S$
       ;
CC 640 NEXT:PRINT"{RVS}";:A=CK
       :GOSUB350:PRINT
KH 650 F=1:AD=AD+8:IF AD>EA TH
       ENPRINT"{DOWN}{BLU}** E
       ND OF DATA **":GOTO220
KC 660 GET A$:IF A$=R$ THEN GO
       SU81080:GOTO220
EQ 670 IF A$=S$ THEN F=P+1:GOS
       UB1080
AD 680 ONFGOTO630,660,630
CM 690 PRINT"{DOWN}{RVS} LOAD
       {SPACE}DATA ":OP=1:GOTO
       710
PC 700 PRINT"{DOWN}{RVS} SAVE
       {SPACE}FILE ":OP=0
RX 710 IN$=N$:INPUT"{DOWN}FILE
       NAME{4}";IN$:IF IN$=N$
       {SPACE}THEN220
PR 720 F=0:PRINT"{DOWN}{BLK}
       {RVS}T{OFF}APE OR {RVS}
       D{OFF}ISK: {4}";
PP 730 GET A$:IF A$="T"THEN PR
       INT"T{DOWN}":GOTO880
HQ 740 IF A$<>"D"THEN730
HH 750 PRINT"D{DOWN}":OPEN15,8
       ,15,"I0:":8=EA-SA:IN$="
       0:"+IN$:IF OP THEN810
SQ 760 OPEN 1,8,8,IN$+",P,W":G
       OSUB860:IF A THEN220
FJ 770 AH=INT(SA/256):AL=SA-(A
       H*256):PRINT#1,CHR$(AL)
       ;CHR$(AH);
PE 780 FOR I=0 TO 8:PRINT#1,CH
       R$(PEEK(8S+I));:IF ST T
       HEN800
FC 790 NEXT:CLOSE1:CLOSE15:GOT
       O940
GS 800 GOSU81060:PRINT"{DOWN}
       {8LK}ERROR DURING SAVE:
       {4}":GOSUB860:GOTO220
MA 810 OPEN 1,8,8,IN$+",P,R":G
       OSU8860:IF A THEN220
GE 820 GET#1,A$,8$:AD=ASC(A$+Z
       $)+256*ASC(8$+Z$):IF AD
       <>SA THEN F=1:GOTO850
RX 830 FOR I=0 TO 8:GET#1,A$:P
       OKE 8S+I,ASC(A$+Z$):IF(
       I<>8)AND ST THEN F=2:AD
       =I:I=8
PA 840 NEXT:IF ST<>64 THEN F=3
FQ 850 CLOSE1:CLOSE15:ON ABS(F
       >0)+1 GOTO960,970
SA 860 INPUT#15,A,A$:IF A THEN
        CLOSE1:CLOSE15:GOSU810
       60:PRINT"{RVS}ERROR: "A
       $
GQ 870 RETURN
EJ 880 POKE183,PEEK(FA+2):POKE
       187,PEEK(FA+3):POKE188,
       PEEK(PA+4):IFOP=0THEN92
       0
HJ 890 SYS 63466:IF(PEEK(783)A
       ND1)THEN GOSUB1060:PRIN
       T"{DOWN}{RVS} FILE NOT
       {SPACE}FOUND ":GOTO690
CS 900 AD=PEEK(829)+256*PEEK(8
       30):IF AD<>SA THEN F=1:
       GOTO970
SC 910 A=PEEK(831)+256*PEEK(83
       2)-1:F=F-2*(A<EA)-3*(A>
       EA):AD=A-AD:GOTO930
KM 920 A=SA:8=EA+1:GOSU81010:P
       OKE780,3:SYS 63338
JF 930 A=BS:8=BS+(EA-SA)+1:GOS
       U81010:ON OP GOTO950:SY
       S 63591
AE 940 GOSU81080:PRINT"{BLU}**
        SAVE COMPLETED **":GOT
       O220
XP 950 POKE147,0:SYS 63562:IF
       {SPACE}ST>0 THEN970
FR 960 GOSU81080:PRINT"{BLU}**
        LOAD COMPLETED **":GOT
       O220
DP 970 GOSUB1060:PRINT"{BLK}
       {RVS}ERROR DURING LOAD:
       {DOWN}{4}":ON F GOSUB98
       0,990,1000:GOTO220
PP 980 PRINT"INCORRECT STARTIN
       G ADDRESS (";:GOSUB8360:
       PRINT")":RETURN
GR 990 PRINT"LOAD ENDED AT ";:
       AD=SA+AD:GOSUB8360:PRINT
       D$:RETURN
FD 1000 PRINT"TRUNCATED AT END
        ING ADDRESS":RETURN
RX 1010 AH=INT(A/256):AL=A-(AH
        *256):POKE193,AL:POKE1
        94,AH
FF 1020 AH=INT(B/256):AL=8-(AH
        *256):POKE174,AL:POKE1
        75,AH:RETURN
FX 1030 IP AD<SA OR AD>EA THEN
        1050
HA 1040 IF(AD>511 AND AD<40960
        )OR(AD>4915I AND AD<53
        248)THEN GOSU81080:F=0
        :RETURN
HC 1050 GOSU81060:PRINT"{RVS}
        {SPACE}INVALID ADDRESS
         {DOWN}{8LK}":F=1:RETU
        RN
AR 1060 POKE SD+5,31:POKE SD+6
        ,208:POKE SD,240:POKE
        {SPACE}SD+1,4:POKE SD+
        4,33
DX 1070 FOR S=1 TO 100:NEXT:GO
        TO1090
PF 1080 POKE SD+5,8:POKE SD+6,
        240:POKE SD,0:POKE SD+
        1,90:POKE SD+4,17
AC 1090 FOR S=1 TO 100:NEXT:PO
        KE SD+4,0:POKE SD,0:PO
        KE SD+1,0:RETURN
```

©

New Products From Batteries Included

An extensive array of new software programs for Apple, Atari, Commodore, and IBM computers has been announced by Batteries Included. The list includes the following:

*I*S Talk* is a full-featured GEM-based telecommunications program for the Atari ST and the IBM PC. The program is part of the company's Integral Solutions line of GEM-based products. *I*S Talk* features a built-in 50,000-word spelling checker, three levels of automation functions, keystroke macros, and an unlimited capture buffer including X-modem file transfers. Other features include Replay, and a unique session recorder. A custom keyboard interface is built in, allowing you to use the mouse or function keys and keyboard.

Suggested retail prices are $79.95 for the ST version and $99.95 for the IBM version.

The popular *HomePak* three-in-one word processor, database manager, and telecommunications package should be available by the time you read this in versions for the Apple Macintosh, the Atari ST, and the Commodore 128 computers. *HomePak* sells for $49.95, and is already available for the Commodore 64, Apple II, IBM PC/PCjr, and Atari XL/XE computers.

The *PaperClip II* word processor for the Commodore 128 is designed to take advantage of the 128's memory, speed, and power. The program includes a built-in telecommunications module for use with online services, a 20,000-word spell checker, macro capability, multiple columns, reverse video scroll, word wrap, chaptering, and an expanded maximum document size. *PaperClip II*, priced at $79.95, is completely compatible with *PaperClip* text files on the Commodore 64. Also new (or planned for release soon) are *PaperClip With SpellPack* for the Atari 130XE ($59.95); *PaperClip* for the Apple II, II+, IIc, and IIe computers ($59.95); and *I*S PaperClip Elite* for the IBM PC and compatibles ($129.95), Atari ST ($79.95), and Commodore Amiga ($129.95). *PaperClip Elite* includes all of the earlier *PaperClip* features, plus advanced functions such as a real-time spelling checker, idea processing, independent linked windows, integrated text and graphics, and other capabilities.

*I*S Degas Elite*, an upgraded version of the original *Degas* graphics and design program for the Atari ST, will be available soon for the IBM PC and compatibles, Amiga, and ST. The program includes everything in the original, plus features like FLIP, SCALE, ROTATE, and the ability to cut and paste between pictures on multiple work screens. *DEGAS Elite* files can also be integrated with *PaperClip Elite* text files, and all *Degas Elite* files are compatible with the original *Degas*.

Batteries Included will also be marketing *The Isgur Portfolio System (IPS)* for the IBM PC ($249.95), the Atari ST ($199.95), the Amiga ($249.95), and the Apple Macintosh ($249.95). This is an investment management program designed by Lee Isgur, Wall Street analyst and first vice president of Paine Webber in New York. *IPS* lets you update your portfolios automatically from online services, and provides a variety of analytical tools for more profitable decision making.

BTS The Spreadsheet is a $69.95 spreadsheet for the Macintosh, Atari ST, and Amiga that includes key math, statistical, and financial functions, logical operators, formatting enhancements, and other features. Maximum worksheet size is 1000 rows by 1000 columns. Desk accessory version is included on the same Macintosh and ST disks.

TimeLink is an electronic diary program for planning and record-keeping, available for the Macintosh and Atari ST for $49.95 each. *I*S Time And Billing* is a professional office administration program planned for the fourth quarter of 1986 for IBM and Atari ST computers. *I*S Consultant*, also planned for the fourth quarter, is an enhanced Atari ST version of the original *Consultant* database. The new program will use the GEM interface and contain additional features. *B/GRAPH Elite* (a fourth quarter release) is a $69.95 graphics/charting and statistical analysis package for the Atari ST.

Soon to be available is *Thunder*, a $39.95 writer's accessory that can be called for use from within any GEM-based Atari ST application. Features include a 50,000-word spelling checker with three different error-catching modes and an immediate word replacement capability. An Abbreviation function will even supply you with the remainder of the word, or words, you want when you type in just the first few letters. And a built-in report analyzer gives you details on everything from word count to the document's relative readability.

Batteries Included, 30 Mural St., Richmond Hill, Ontario, Canada L4B 1B5.
Circle Reader Service Number 200.

Abacus Software, Book

Abacus Software has introduced several Atari ST software applications packages, each priced at $39.95. *ST TextPro* is a word processor that features multicolumn output, automatic indexing and table of contents, scrolling, definable function keys, sideways output (to Epson printers), and printer drivers for other printers. The program allows full-screen editing with either mouse or keyboard commands.

ST Text Designer is a pagemaking package for creating layouts from word processing files. The program reads text files from *ST TextPro* and other ASCII word processors, and allows block operations. Graphics can be merged into the layout, and borders and lines can be added. Output is to Epson-compatible printers.

ST DataPro is a data management package that lets you input data through screen templates. Record length is unlimited, with a maximum of 64,000 records available. The package also supports a RAM disk as well as a floppy disk. *ST Forth/MT* is a multitasking implementation of the Forth programming language, based on Forth 83. The program supports 32-bit arithmetic, has more than 1500 commands, and includes a full-screen editor, Forth macro assembler, and monitor and disk monitor.

Abacus is also announcing *ST PaintPro*, a drawing program, and *ST AssemPro*, an assembly language development package.

For the Commodore 128 computer, Abacus recently released another volume in its Commodore 128 Reference Library. The $19.95 title is *Commodore 1571 Internals*.

Abacus Software, P.O. Box 7211, Grand Rapids, MI 49510.

Circle Reader Service Number 201.

64 Software From Firebird

Firebird has released several new software packages for the Commodore 64 and 128.

The Music System is a comprehensive music development package that includes full sonic tailoring of the computer's SID chip, multi-voicing, mono or polyphonic modes, editing, recording, and storage and playback of sound settings and compositions.

There is an advanced version of *The Music System* which includes the features of the standard version and adds MIDI capabilities (compatible with either S.I.E.L. or Passport Designs MIDI interfaces). The advanced version enables the user to link and edit sequences, control six MIDI tracks or devices simultaneously, print sheet music, and also permits automatic transpositions and automatic tempo conforming and correcting. You can upgrade the standard version to the advanced version at a nominal cost.

Gerry The Germ and *Microcosm* are the newest in the Firebird Super Silver Disk Series. Both games feature music, sound effects, graphics, and animation. *Gerry The Germ* is a journey through the human body with Gerry as the guide. In *Microcosm*, the player must defend the agricultural cargo of a crippled interstellar freightliner against a hoard of mutant insects. The games are on one *flippy* disk and require either joystick or keyboard control.

The standard version of *The Music System* has a suggested retail price of $39.95 and the advanced version is $79.95. The *Gerry The Germ/Microcosm* Super Silver Disk Series package retails for $19.95.

Firebird, P.O. Box 49, Ramsey, NJ 07446.

Circle Reader Service Number 202.

New Games From Intellicreations

Intellicreations (formerly Datasoft) has announced the release of three new games.

The Never Ending Story is an adventure/fantasy game based on the book and film of the same name. Cast as the hero, Atreyu, and aided by Falkor, the luck dragon, you face the trials and terrors of the ever-consuming "nothing" in an illustrated and imaginative computer adventure. *The Never Ending Story* is now available as a *flippy* for Commodore and Atari 8-bit machines (Commodore version on one side, Atari on the flip side), and for the Apple II computers. All versions retail for $29.95.

Mind Pursuit is a test of intelligence, knowledge, and trivia with three levels of difficulty. At the simplest level are true/false questions; next are multiple choice; and finally, and most difficult, are fill-in-the-blanks. The game is designed with questions for both adults and children, and has three difficulty levels, so the whole family can play together. Music and graphics clues are also used for variety, and provide further challenges to the game play. Additional game disks are available. Commodore and Apple versions retail for $29.95; add-on disks are $14.95.

221B Baker Street is the address of fictional super-sleuth Sherlock Holmes. In the game *221B Baker Street*, you start at that address and travel through the streets and alleys of London, gathering clues that will lead to the solution of some of the most intriguing cases ever faced by Holmes and his assistant, Dr. Watson. The initial game will include 30 different cases; two additional game disks will be available later, each containing 30 more cases. Atari, Commodore, and Apple versions retail for $29.95, with follow-up disks costing $14.95 each.

Intellicreations, 19808 Nordhoff Pl., Chatsworth, CA 91311.

Circle Reader Service Number 203.

128 Telecommunications And 64 Graphics

Progressive Peripherals & Software has announced two new products for the Commodore 64 and 128.

For the Commodore 128, *BobsTerm Pro-128* is a menu-driven communications package that lets you edit files while it reads, writes, uploads, and downloads to any disk type (including CP/M). The package supports VT-100 and VT-52 80 ADM-3I (CP/M type) terminal emulation, and includes a full-screen text editor, on-screen status display, remote mode, macro and answerback string functions, and a manual. The program is compatible with most modems and works with high speed 1571 disk drives, the SFD-1001, and Commodore and MSD dual drives.

With *Picasso's Revenge*, you can draw circles, squares, triangles, and many other geometric figures with the Commodore 64. This graphics package has 35 predefined textures, five-level focus, and a ZOOM command that magnifies the screen area eight times. The program supports high-resolution drawings made with Koala, Suncom, Paint Majic, and other commonly used graphics programs. It's compatible with most dot-matrix printers and prints in nine shades of gray. The *Picasso's Revenge* package includes a free light pen and a user's manual.

BobsTerm Pro-128 retails for $79.95 and *Picasso's Revenge* for $59.95.

Progressive Peripherals & Software, 464 Kalamath St., Denver, CO 80204.

Circle Reader Service Number 204.

Amiga Fonts

Classic Concepts FUTUREWARE has announced new type fonts for Amiga users who need larger fonts for video titling, graphics, desktop publishing, and other applications. *FUTUREWARE FONTS* can be used to mix titles with video images (with Genlock) and are more clearly legible in high resolution mode than fonts with the *Workbench* disk. The fonts are compatible with *Notepad, Deluxe Paint, Aegis Images*, and other common Amiga software

Each package includes a disk utility installation program, 13 new fonts, and a font reference booklet.

Retail price is $14.95, plus $1 shipping and handling.

FUTUREWARE FONTS, P.O. Box 94276, Richmond, B.C., Canada V6Y 2A6.

Circle Reader Service Number 205.

Apple II Science Software

Science Toolkit—Master Module lets you perform real experiments with your Apple computer and two sensory probes. Each package contains an interface box, light probe, temperature probe, light probe stand, light guard, two lab notebook labels, and a 125-page *User's Manual & Experiment Guide* that details how to use the probes and the on-screen instruments—thermometer, light meter, timer, and strip chart.

Science Toolkit—Master Module is for the Apple IIe and IIc with 64K memory. Printer is optional and the interface box is included. Users should be 12 years old and up.

There is a school version which includes a teacher's guide in addition to the material provided in the standard package. For use in grades 4–12.

The standard version retails for $69.95 and the school version retails for $89.95. Anyone with the standard version can buy the teacher's guide separately for $20.

Brøderbund Software, Inc., 17 Paul Dr., San Rafael, CA 94903-2101.

Circle Reader Service Number 206.

©

From the publishers of *COMPUTE!*

July 1986 *COMPUTE!* Disk

All the exciting programs from the past three issues of *COMPUTE!* are on timesaving, error-free floppy disks that are ready to load on your IBM PC and PCjr or Commodore 64 and 128. The July 1986 *COMPUTE! Disks* contain the entertaining and useful Commodore or IBM programs from the May, June, and July 1986 issues of *COMPUTE!*.

The July 1986 *COMPUTE! Disk* costs $12.95 plus $2.00 shipping and handling and is available only from COMPUTE! Publications. Please specify whether you need a Commodore or IBM disk.

For added savings and convenience, you may also subscribe to the *COMPUTE! Disk*. At a cost of only $39.95 a year (a $12.00 savings), you'll receive four disks, one every three months. Each disk will contain all the programs for your machine from the previous three issues of *COMPUTE!*.

This is an excellent way to build your software library while you enjoy the quality programs from *COMPUTE!*.

Disks and subscriptions are available for Apple, Atari, Commodore 64 and 128, and IBM personal computers. Call for details.

For more information or to order the July 1986 *COMPUTE! Disk*, call toll free 1-800-346-6767 (in NY 212-265-8360) or write *COMPUTE! Disk*, P.O. Box 5038, F.D.R. Station, New York, NY 10150.

COMPUTE! Publications, Inc. abc
Part of ABC Consumer Magazines, Inc.
One of the ABC Publishing Companies
825 7th Avenue, 6th Floor, New York, NY 10019
Publishers of COMPUTE!, COMPUTE!'s Gazette, COMPUTE!'s Gazette Disk, COMPUTE! Books, and COMPUTE!'s Apple Applications

Atari COMPUTE! DISK

The first quarterly COMPUTE! DISK for Atari, containing the programs from the January–March 1986 issues, has caused some confusion among those who received it. First, the disk *does not* contain a disk operating system (DOS) and therefore cannot be used to start up the computer. As the disk label instructs, you must boot your system using your own DOS disk, make sure you are in BASIC, then put the COMPUTE! DISK in the drive and type RUN "D:MENU". The disk is in Atari DOS 2.0S format, which can also be read by Atari DOS 2.5 and most third-party operating systems such as DOS XL and OS/A+. Atari DOS 3.0 has an option to convert DOS 2.0 files to 3.0 format. However, if you're still using DOS 3.0 you should seriously consider switching to DOS 2.5 (despite the confusing numbering, version 2.5 is newer than 3.0). DOS 2.5 is available free from many Atari dealers and user groups. It is not possible to write DOS files to the COMPUTE! DISK because it is write protected and because there isn't enough room on the disk for any more files.

Note, however, that while the COMPUTE! DISK is write protected, it is not copy protected. When you receive the disk, you should first make a backup copy. (With Atari DOS 2.0 or 2.5, use option J—Duplicate Disk). Programs such as *SpeedScript* and *SpeedCalc* will be more useful if copied to a disk that contains the DOS system files. (With Atari DOS 2.0 or 2.5, use option O—Duplicate File).

There is one program on the January–March Atari disk that cannot be used as-is with Atari DOS. "High Rise" was designed to be a boot disk file if typed in from the magazine using MLX. The verison on the disk was intended to be executed as a binary file, but we failed to explain that and also failed to put the program on the disk in the proper format for the Atari DOS L (Load Binary File) option. The problem is easy to correct, though. Copy the HIRISE.FEB program from the COMPUTE! DISK to another disk (preferably one that contains DOS) using the O option in Atari DOS 2.0/2.5, then enter and run the following one-line program:

```
10 OPEN #1,9,0,"D:HIRISE.
   FEB":PUT #1,224:PUT #1
   ,2:PUT #1,225:PUT #1,2
   :PUT #1,0:PUT #1,4B:CL
   OSE #1
```

This fixes the file so the game can be played by going to the Atari DOS 2.0/2.5 menu and using the L option. Specify HIRISE.FEB as the file to load; it begins running automatically after loading. You could also use the DOS menu's E option to rename the fixed file to AUTORUN.SYS. In that case, the program will load and run automatically when the disk is booted. If you're using OS/A+ DOS or DOS XL, you can run High Rise simply by typing HIRISE.FEB at a DOS D1: prompt, so the above correction is not necessary.

The January–March Atari disk does not contain the programs "MLX" and "Automatic Proofreader" because the disk was too full for them to be included. They appear on the April–June Atari disk and should be on all future Atari disks when space allows. We know of no special problems with any of the programs on the April–June disk.

64 Autobooter

The automatic loading technique used in this program from the May issue (p. 92) should work fine for the first program autobooted after the computer is turned on. However, when autobooting more than one program in a session, the routine may fail if a later autoboot attempts to load a program shorter than the one previously autobooted. To correct this, change the following lines:

```
GA 140 IF S<>51988 THEN PRINT:
       PRINT"MISTYPED DATA":EN
       D
DF 200 IF B>7936 THEN PRINT CH
       R$(18);STR$(B-7936)"
       {2 SPACES}BYTE OVERFLOW
        ":GOTO 160
CG 550 DATA 57,224,32,176,12,1
       89,16,255
BE 710 DATA 104,208,11,32,51,1
       65,162,54
AS 720 DATA 160,2,134,122,132,
       123,162,31
EF 730 DATA 169,0,157,16,245,2
       02,16,250
RM 740 DATA 169,13,76,210,255,
       138,0,0
```

Minding IBM Memory

Line 100 in Program 3 (p. 86) of this memory management utility in the June issue contains an error that prevents the deallocation routine from properly releasing allocated blocks of memory. The &h85 in that line should be &h8b, as shown in the assembly listing for the deallocation routine (Program 2).

Using PALETTE USING

In Program 2 from this article in the May issue (p. 71), the AS$="N" at the end of line 80 should instead be AN$="N".

Commodore Loading and Linking, Part 4

In the example program in the lower-right corner of page 75 in the June issue, omit the question mark in the value 128 in line 140. The question mark was caused by a printer malfunction.

COMPUTE!'s Guide To Typing In Programs

Computers are precise—type the program *exactly* as listed, including necessary punctuation and symbols, except for special characters noted below. We have provided a special listing convention as well as a program to check your typing—"The Automatic Proofreader."

Programs for the IBM, TI-99/4A, and Atari ST models should be typed exactly as listed; no special characters are used. Programs for Commodore, Apple, and Atari 400/800/XL/XE computers may contain some hard-to-read special characters, so we have a listing system that indicates these control characters. You will find these Commodore and Atari characters in curly braces; *do not type the braces.* For example, {CLEAR} or {CLR} instructs you to insert the symbol which clears the screen on the Atari or Commodore machines. A complete list of these symbols is shown in the tables below. For Commodore, Apple, and Atari, a single symbol by itself within curly braces is usually a control key or graphics key. If you see {A}, hold down the CONTROL key and press A. This will produce a reverse video character on the Commodore (in quote mode), a graphics character on the Atari, and an invisible control character on the Apple.

Graphics characters entered with the Commodore logo key are enclosed in a special bracket: [<A>]. In this case, you would hold down the Commodore logo key as you type A. Our Commodore listings are in uppercase, so shifted symbols are underlined. A graphics heart symbol (SHIFT-S) would be listed as S. One exception is {SHIFT-SPACE}. When you see this, hold down SHIFT and press the space bar. If a number precedes a symbol, such as {5 RIGHT}, {6 S}, or [<8 Q>], you would enter five cursor rights, six shifted S's, or eight Commodore-Q's. On the Atari, inverse characters (white on black) should be entered with the inverse video

Atari 400/800/XL/XE

When you see	Type	See	
{CLEAR}	ESC SHIFT <		Clear Screen
{UP}	ESC CTRL -	↑	Cursor Up
{DOWN}	ESC CTRL =	↓	Cursor Down
{LEFT}	ESC CTRL +	←	Cursor Left
{RIGHT}	ESC CTRL *	→	Cursor Right
{BACK S}	ESC DELETE	◄	Backspace
{DELETE}	ESC CTRL DELETE		Delete character
{INSERT}	ESC CTRL INSERT		Insert character
{DEL LINE}	ESC SHIFT DELETE		Delete line
{INS LINE}	ESC SHIFT INSERT		Insert line
{TAB}	ESC TAB	►	TAB key
{CLR TAB}	ESC CTRL TAB		Clear tab
{SET TAB}	ESC SHIFT TAB		Set tab stop
{BELL}	ESC CTRL 2		Ring buzzer
{ESC}	ESC ESC		ESCape key

Commodore PET/CBM/VIC/64/128/16/+4

When You Read:	Press:	See:	When You Read:	Press:	See:
{CLR}	SHIFT CLR/HOME		[<1>]	COMMODORE 1	
{HOME}	CLR/HOME		[<2>]	COMMODORE 2	
{UP}	SHIFT ↑ CRSR ↓		[<3>]	COMMODORE 3	
{DOWN}	↑ CRSR ↓		[<4>]	COMMODORE 4	
{LEFT}	SHIFT ← CRSR →		[<5>]	COMMODORE 5	
{RIGHT}	← CRSR →		[<6>]	COMMODORE 6	
{RVS}	CTRL 9		[<7>]	COMMODORE 7	
{OFF}	CTRL 0		[<8>]	COMMODORE 8	
{BLK}	CTRL 1		{ F1 }	f1	
{WHT}	CTRL 2		{ F2 }	SHIFT f1	
{RED}	CTRL 3		{ F3 }	f3	
{CYN}	CTRL 4		{ F4 }	SHIFT f3	
{PUR}	CTRL 5		{ F5 }	f5	
{GRN}	CTRL 6		{ F6 }	SHIFT f5	
{BLU}	CTRL 7		{ F7 }	f7	
{YEL}	CTRL 8		{ F8 }	SHIFT f7	
			←	←	

```
180 IF VAL(LEFT$(L$,2))=Ø AND
    M1D$(L$,3,1)=" " THEN L$=M
    ID$(L$,4)
200 IF ASC(L$)>57 THEN 26Ø 'no
     line number, therefore co
    mmand
205 BL=INSTR(L$," "):IF BL=Ø T
    HEN BL$=L$:GDTO 206 ELSE B
    L$=LEFT$(L$,BL-1)
206 LNUM=VAL(BL$):TEXT$=M1D$(L
    $,LEN(STR$(LNUM))+1)
210 IF TEXT$="" THEN GDSUB 54Ø
    :IF LNUM=LNUM(P) THEN GDSU
    B 56Ø:GDTD 15Ø ELSE 15Ø
220 CK5UM=Ø:FOR I=1 TD LEN(L$)
    :CKSUM=(CK5UM+A5C(MID$(L$,
    1))*1) AND 255:NEXT:LOCATE
     Y,1:PRINT CHR$(65+CKSUM/1
    6)+CHR$(65+(CKSUM AND 15))
    +" "+L$
230 GOSUB 54Ø:IF LNUM(P)=LNUM
    THEN L$(P)=TEXT$:GOTO 15Ø
    'replace line
240 GO5UB 58Ø:GOTD 15Ø 'insert
     the line
260 TEXT$="":FOR 1=1 TO LEN(L$
    ):A=ASC(MID$(L$,I)):TEXT$=
    TEXT$+CHR$(A+32*(A>96 AND
    A<123)):NEXT
270 DEL1MITER=1NSTR(TEXT$," ")
    :COMMAND$=TEXT$:ARG$="":IF
     DELIM1TER THEN COMMAND$=L
    EFT$(TEXT$,DELIM1TER-1):AR
    G$=MID$(TEXT$,DELIMITER+1)
     ELSE DELIMITER=IN5TR(TEXT
    $,CHR$(34)):IF DEL1MITER T
    HEN CDMMAND$=LEFT$(TEXT$,D
    ELIMITER-1):ARG$=MID$(TEXT
    $,DELIMITER)
280 IF CDMMAND$<>"LIST" THEN 4
    1Ø
290 DPEN "scrn:" FOR OUTPUT A5
     #1
300 IF ARG$="" THEN F1RST=Ø:P=
    MAX-1:GDTO 34Ø
310 DELIMITER=INSTR(ARG$,"-"):
    IF DELIMITER=Ø THEN LNUM=V
    AL(ARG$):GOSUB 54Ø:FIRST=P
    :GOTO 34Ø
320 FIRST=VAL(LEFT$(ARG$,DELIM
    1TER)):LAST=VAL(MID$(ARG$,
    DELIMITER+1))
330 LNUM=FIRST:GDSUB 54Ø:FIRST
    =P:LNUM=LAST:GOSUB 54Ø:IF
    P=Ø THEN P=MAX-1
340 FOR X=FIRST TD P:N$=MID$(S
    TR$(LNUM(X)),2)+" "
350 IF CKFLAG=Ø THEN A$="":GDT
    O 37Ø
360 CKSUM=Ø:A$=N$+L$(X):FOR I=
    1 TD LEN(A$):CKSUM=(CKSUM+
    ASC(MID$(A$,I))*I) AND 255
    :NEXT:A$=CHR$(65+CKSUM/16)
    +CHR$(65+(CKSUM AND 15))+"
     "
370 PR1NT #1,A$+N$+L$(X)
380 IF INKEY$<>"" THEN X=P
390 NEXT :CLOSE #1:CKFLAG=Ø
400 GOTO 13Ø
410 IF CDMMAND$="LLIST" THEN O
    PEN "lpt1:" FDR OUTPUT AS
    #1:GDTD 3ØØ
420 IF COMMAND$="CHECK" THEN C
    KFLAG=1:GOTO 29Ø
430 IF COMMAND$<>"SAVE" THEN 4
    5Ø
440 GOSUB 6ØØ:OPEN ARG$ FOR OU
    TPUT AS #1:ARG$="":GOTO 3Ø
    Ø
450 IF COMMAND$<>"LDAO" THEN 4
    9Ø
460 GOSUB 6ØØ:OPEN ARG$ FDR 1N
    PUT AS #1:MAX=Ø:P=Ø
470 WH1LE NOT EOF(1):LINE INPU
    T #1,L$:BL=INSTR(L$," "):B
    L$=LEFT$(L$,BL-1):LNUM(P)=
    VAL(BL$):L$(P)=M1D$(L$,LEN
    (STR$(VAL(BL$)))+1):P=P+1:
    WEND
480 MAX=P:CLDSE #1:GDTD 13Ø
490 1F COMMAND$="NEW" THEN 1NP
    UT "Erase program - Are yo
    u sure";L$:IF LEFT$(L$,1)=
    "y" OR LEFT$(L$,1)="Y" THE
    N MAX=Ø:LNUM(Ø)=65536!:GDT
    O 13Ø:ELSE 13Ø
500 1F CDMMAND$="BASIC" THEN C
    OLOR 7,Ø,Ø:DN ERRDR GDTD Ø
    :CLS:END
510 1F COMMAND$<>"F1LES" THEN
    52Ø
515 IF ARG$="" THEN ARG$="A:"
    ELSE SEL=1:GOSUB 6ØØ
517 FILES ARG$:GOTD 13Ø
520 PRINT"Syntax error":GDTO 1
    3Ø
540 P=Ø:WHILE LNUM>LNUM(P) AND
     P<MAX:P=P+1:WEND:RETURN
560 MAX=MAX-1:FOR X=P TO MAX:L
    NUM(X)=LNUM(X+1):L$(X)=L$(
    X+1):NEXT:RETURN
580 MAX=MAX+1:FOR X=MAX TO P+1
     STEP -1:LNUM(X)=LNUM(X-1)
    :L$(X)=L$(X-1):NEXT:L$(P)=
    TEXT$:LNUM(P)=LNUM:RETURN
600 IF LEFT$(ARG$,1)<>CHR$(34)
     THEN 52Ø ELSE ARG$=MIO$(A
    RG$,2)
610 IF RIGHT$(ARG$,1)=CHR$(34)
     THEN ARG$=LEFT$(ARG$,LEN(
    ARG$)-1)
620 IF SEL=Ø AND INSTR(ARG$,".
    ")=Ø THEN ARG$=ARG$+".BAS"
630 SEL=Ø:RETURN
640 CLOSE #1:CKFLAG=Ø:PRINT"St
    opped.":RETURN 15Ø
650 PRINT "Error #";ERR:RESUME
     15Ø
```

Program 3: Commodore Proofreader

By Philip Nelson, Assistant Editor

```
10 VEC=PEEK(772)+256*PEEK(773)
   :LO=43:HI=44
20 PRINT "AUTOMATIC PROOFREADE
   R POR ";:IF VEC=42364 THEN
   {SPACE}PRINT "C-64"
30 IF VEC=5Ø556 THEN PRINT "VI
   C-2Ø"
40 IF VEC=35158 THEN GRAPHIC C
   LR:PRINT "PLUS/4 & 16"
50 IF VEC=17165 THEN LO=45:HI=
   46:GRAPHIC CLR:PRINT"128"
60 SA=(PEEK(LO)+256*PEEK(HI))+
   6:ADR=SA
70 FOR J=Ø TO 166:READ BYT:POK
   E ADR,BYT:ADR=ADR+1:CHK=CHK
   +BYT:NEXT
80 IF CHK<>2Ø57Ø THEN PRINT "*
    ERROR* CHECK TYPING IN DATA
    STATEMENTS":END
90 FOR J=1 TO 5:READ RF,LF,HF:
   RS=SA+RP:HB=INT(RS/256):LB=
   RS-(256*HB)
100 CHK=CHK+RF+LF+HF:POKE SA+L
    F,LB:POKE SA+HF,HB:NEXT
110 IP CHK<>22Ø54 THEN PRINT "
    *ERROR* RELOAD PROGRAM AND
    {SPACE}CHECK PINAL LINE":EN
    D
120 POKE SA+149,PEEK(772):POKE
     SA+15Ø,PEEK(773)
130 IF VEC=17165 THEN POKE SA+
    14,22:POKE SA+18,23:POKESA+
    29,224:POKESA+139,224
140 PRINT CHR$(147);CHR$(17);"
    PROOFREADER ACTIVE":SYS SA
150 POKE HI,PEEK(HI)+1:POKE (P
    EEK(LO)+256*PEEK(HI))-1,Ø:N
    EW
160 DATA 12Ø,169,73,141,4,3,16
    9,3,141,5,3
170 DATA 88,96,165,2Ø,133,167,
    165,21,133,168,169
180 DATA Ø,141,Ø,255,162,31,18
    1,199,157,227,3
190 DATA 2Ø2,16,248,169,19,32,
    21Ø,255,169,18,32
200 DATA 21Ø,255,16Ø,Ø,132,18Ø
    ,132,176,136,23Ø,18Ø
210 DATA 2ØØ,185,Ø,2,24Ø,46,2Ø
    1,34,2Ø8,8,72
220 DATA 165,176,73,255,133,17
    6,1Ø4,72,2Ø1,32,2Ø8
230 DATA 7,165,176,2Ø8,3,1Ø4,2
    Ø8,226,1Ø4,166,18Ø
240 DATA 24,165,167,121,Ø,2,13
    3,167,165,168,1Ø5
250 DATA Ø,133,168,2Ø2,2Ø8,239
    ,24Ø,2Ø2,165,167,69
260 DATA 168,72,41,15,168,185,
    211,3,32,21Ø,255
270 DATA 1Ø4,74,74,74,74,168,1
    85,211,3,32,21Ø
280 DATA 255,162,31,189,227,3,
    149,199,2Ø2,16,248
290 DATA 169,146,32,21Ø,255,76
    ,86,137,65,66,67
300 DATA 68,69,7Ø,71,72,74,75,
    77,8Ø,81,82,83,88
310 DATA 13,2,7,167,31,32,151,
    116,117,151,128,129,167,136
    ,137
```

Program 4: Apple Proofreader

By Tim Victor, Editorial Programmer

```
10 C = Ø: FOR I = 768 TO 768 +
   68: READ A:C = C + A: POKE I
   ,A: NEXT
20 IF C < > 7258 THEN PRINT "ER
   RDR IN PROOFREADER DATA STAT
   EMENTS": END
30 IF PEEK (19Ø * 256) < > 76 T
   HEN POKE 56,Ø: POKE 57,3: CA
   LL 1ØØ2: GOTD 5Ø
40 PRINT CHR$ (4);"IN#A$3ØØ"
50 PDKE 34,Ø: HOME : POKE 34,1:
    VTAB 2: PRINT "PROOFREADER
   INSTALLED"
60 NEW
100 DATA 216,32,27,253,2Ø1,141
110 DATA 2Ø8,6Ø,138,72,169,Ø
120 DATA 72,189,255,1,2Ø1,16Ø
130 DATA 24Ø,8,1Ø4,1Ø,125,255
140 DATA 1,1Ø5,Ø,72,2Ø2,2Ø8
150 DATA 238,1Ø4,17Ø,41,15,9
160 DATA 48,2Ø1,58,144,2,233
170 DATA 57,141,1,4,138,74
180 DATA 74,74,74,41,15,9
190 DATA 48,2Ø1,58,144,2,233
200 DATA 57,141,Ø,4,1Ø4,17Ø
210 DATA 169,141,96
```

©

key (Atari logo key on 400/800 models).

Whenever more than two spaces appear in a row, they are listed in a special format. For example, {6 SPACES} means press the space bar six times. Our Commodore listings never leave a single space at the end of a line, instead moving it to the next printed line as {SPACE}.

Amiga program listings contain only one special character, the left arrow (←) symbol. This character marks the end of each program line. Wherever you see a left arrow, press RETURN or move the cursor off the line to enter that line into memory. Don't try to type in the left arrow symbol; it's there only as a marker to indicate where each program line ends.

The Automatic Proofreader

Type in the appropriate program listed below, then save it for future use. The Commodore Proofreader works on the Commodore 128, 64, Plus/4, 16, and VIC-20. Don't omit any lines, even if they contain unfamiliar commands or you think they don't apply to your computer. When you run the program, it installs a machine language program in memory and erases its BASIC portion automatically (so be sure to save several copies before running the program for the first time). If you're using a Commodore 128, Plus/4 or 16, do *not* use any GRAPHIC commands while the Proofreader is active. You should disable the Commodore Proofreader before running any other program. To do this, either turn the computer off and on or enter SYS 64738 (for the 64), SYS 65341 (128), SYS 64802 (VIC-20), or SYS 65526 (Plus/4 or 16). To reenable the Proofreader, reload the program and run it as usual. Unlike the original VIC/64 Proofreader, this version works the same with disk or tape.

On the Atari, run the Proofreader to activate it (the Proofreader remains active in memory as a machine language program); you must then enter NEW to erase the BASIC loader. Pressing SYSTEM RESET deactivates the Atari Proofreader; enter PRINT USR(1536) to reenable it.

The Apple Proofreader erases the BASIC portion of itself after you run it, leaving only the machine language portion in memory. It works with either DOS 3.3 or ProDOS. Disable the Apple Proofreader by pressing CTRL-RESET before running another BASIC program.

The IBM Proofreader is a BASIC program that simulates the IBM BASIC line editor, letting you enter, edit, list, save, and load programs that you type. Type RUN to activate. Be sure to leave Caps Lock on, except when typing lowercase characters.

Once the Proofreader is active, try typing in a line. As soon as you press RETURN, either a hexadecimal number (on the Apple) or a pair of letters (on the Commodore, Atari, or IBM) appears. The number or pair of letters is called a *checksum*.

Compare the value displayed on the screen by the Proofreader with the checksum printed in the program listing in the magazine. The checksum is given to the left of each line number. Just type in the program a line at a time (without the printed checksum), press RETURN or Enter, and compare the checksums. If they match, go on to the next line. If not, check your typing; you've made a mistake. Because of the checksum method used, do not type abbreviations, such as ? for PRINT. On the Atari and Apple Proofreaders, spaces are not counted as part of the checksum, so be sure you type the right number of spaces between quote marks. The Atari Proofreader does not check to see that you've typed the characters in the right order, so if characters are transposed, the checksum still matches the listing. The Commodore Proofreader catches transposition errors and ignores spaces unless they're enclosed in quotation marks. The IBM Proofreader detects errors in spacing and transposition.

IBM Proofreader Commands

Since the IBM Proofreader replaces the computer's normal BASIC line editor, it has to include many of the direct-mode IBM BASIC commands. The syntax is identical to IBM BASIC. Commands simulated are LIST, LLIST, NEW, FILES, SAVE, and LOAD. When listing your program, press any key (except Ctrl-Break) to stop the listing. If you enter NEW, the Proofreader prompts you to press Y to be especially sure you mean yes.

Two new commands are BASIC and CHECK. BASIC exits the Proofreader back to IBM BASIC, leaving the Proofreader in memory. CHECK works just like LIST, but shows the checksums along with the listing. After you have typed in a program, save it to disk. Then exit the Proofreader with the BASIC command, and load the program as usual (this replaces the Proofreader in memory). You can now run the program, but you may want to resave it to disk. This will shorten it on disk and make it load faster, but it can no longer be edited with the Proofreader. If you want to convert an existing BASIC program to Proofreader format, save it to disk with SAVE *"filename"*,A.

Program 1: Atari Proofreader

By Charles Brannon, Program Editor

```
100 GRAPHICS Ø
110 FOR I=1536 TO 17ØØ:REA
    D A:POKE I,A:CK=CK+A:N
    EXT I
12Ø IF CK<>19Ø72 THEN ? "E
    rror in DATA Statement
    s.  Check Typing.":END

13Ø A=USR(1536)
14Ø ? :? "Automatic Proofr
    eader Now Activated."
15Ø END
16Ø DATA 1Ø4,16Ø,Ø,185,26,
    3,2Ø1,69,24Ø,7
17Ø DATA 2ØØ,2ØØ,192,34,2Ø
    8,243,96,2ØØ,169,74
18Ø DATA 153,26,3,2ØØ,169,
    6,153,26,3,162
19Ø DATA Ø,189,Ø,228,157,7
    4,6,232,224,16
2ØØ DATA 2Ø8,245,169,93,14
    1,78,6,169,6,141
21Ø DATA 79,6,24,173,4,228
    ,1Ø5,1,141,95
22Ø DATA 6,173,5,228,1Ø5,Ø
    ,141,96,6,169
23Ø DATA Ø,133,2Ø3,96,247,
    238,125,241,93,6
24Ø DATA 244,241,115,241,1
    24,241,76,2Ø5,238
25Ø DATA Ø,Ø,Ø,Ø,Ø,32,62,2
    46,8,2Ø1
26Ø DATA 155,24Ø,13,2Ø1,32
    ,24Ø,7,72,24,1Ø1
27Ø DATA 2Ø3,133,2Ø3,1Ø4,4
    Ø,96,72,152,72,138
28Ø DATA 72,16Ø,Ø,169,128,
    145,88,2ØØ,192,4Ø
29Ø DATA 2Ø8,249,165,2Ø3,7
    4,74,74,74,24,1Ø5
3ØØ DATA 161,16Ø,3,145,88,
    165,2Ø3,41,15,24
31Ø DATA 1Ø5,161,2ØØ,145,8
    8,169,Ø,133,2Ø3,1Ø4
32Ø DATA 17Ø,1Ø4,168,1Ø4,4
    Ø,96
```

Program 2: IBM Proofreader

By Charles Brannon, Program Editor

```
1Ø 'Automatic Proofreader Vers
   ion 3.Ø (Lines 2Ø5,2Ø6 adde
   d/19Ø deleted/47Ø,49Ø chang
   ed from V2.Ø)
1ØØ DIM L$(5ØØ),LNUM(5ØØ):COLO
    R Ø,7,7:KEY OFF:CLS:MAX=Ø:
    LNUM(Ø)=65536!
11Ø ON ERROR GOTO 12Ø:KEY 15,C
    HR$(4)+CHR$(7Ø):ON KEY(15)
     GOSUB 64Ø:KEY (15) ON:GOT
    O 13Ø
12Ø RESUME 13Ø
13Ø DEF SEG=&H4Ø:W=PEEK(&H4A)
14Ø ON ERROR GOTO 65Ø:PRINT:PR
    INT"Proofreader Ready."
15Ø LINE INPUT L$:Y=CSRLIN-INT
    (LEN(L$)/W)-1:LOCATE Y,1
16Ø DEF SEG=Ø:POKE 1Ø5Ø,3Ø:POK
    E 1Ø52,34:POKE 1Ø54,Ø:POKE
     1Ø55,79:POKE 1Ø56,13:POKE
     1Ø57,28:LINE INPUT L$:DEF
     SEG:IF L$="" THEN 15Ø
17Ø IF LEFT$(L$,1)=" " THEN L$
    =MID$(L$,2):GOTO 17Ø
```

Expert Back-up for Atari® Users.

SPECIALIZING IN BACKUP HARDWARE & SOFTWARE

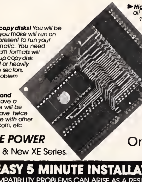
THE 1050 DUPLICATOR™: The most powerful diskdrive copy system ever developed for the ATARI.

The only Copy system You will ever need.

What will it do?

▶ **The main purpose of the Duplicator is to copy disks!** You will be able to copy just about any disk! The copies you make will run on any Atari drive. The Duplicator need not be present to run your backup copies. The Duplicator is fully automatic. You need only insert source and destination disks. Custom formats will be read and in turn reproduced on the backup copy disk. Our device will reproduce any custom format or heavily copy guarded scheme, bad sectors, double sectors, 19 through 24 sector format will present no problem to the Duplicator.

▶ **You will still have single density, density and one half, and double density.** When you have a Duplicator installed in a 1050 drive that drive will be turned into true double density. You will have twice the disk storage. Your drive will be compatible with other double density drives as The Rana Indus Percom, etc.



▶ **High speed read & write.** Your disk drive will read and load all of your software, saving wear and tear on your drive. The 1050 drive now reads one sector at a time. This is slow and inefficient. With the duplicator installed you will be able to read eighteen sectors in the time it takes standard, unenhanced drives to read one.

▶ **Included with every Duplicator will be user friendly disk software.** A simple, menu driven program will allow you to copy all of your software. A Duplicator enhanced drive will be a SMART drive. We plan to write many new and exciting programs that can only be run on an enhanced drive, eg. sending a copy-guarded disk over the phone. Since the drive is now fully programmable, future upgrades can be made available to you on disks, should the need arise. No further hardware changes will ever be needed. The Duplicator comes with a full hardware and software guarantee.

HARDWARE POWER

Fully Compatible with the XL & New XE Series.

Only **$149.95*** Plus $3.50 for Shipping & Handling. Add 7% outside U.S.A. N.Y. State Residents add 7½% Sales Tax.

*Duplicator price may increase due to component shortage and price increases.

*Dealer inquiries are welcome: call for quantity price quote.

EASY 5 MINUTE INSTALLATION

NO HARM TO YOUR DRIVE OR INCOMPATIBILITY PROBLEMS CAN ARISE AS A RESULT OF THE INSTALLATION OF OUR DUPLICATOR.

520 ST Copy Program

THE ST DUPLICATOR™

Our famous 1050 DUPLICATOR has been converted to service the **ATARI™** 520 ST. It is a software only disc copy system.

You can now back up heavily copy-guarded and protected discs. Presently, we can't find anything the ST DUPLICATOR will not copy. And if new forms of software protection should appear on the market, we will provide software upgrades.

ST DUPLICATOR™ INCLUDES USER FRIENDLY SOFTWARE AND INSTRUCTIONS

$59.95 Plus $3.50 Shipping. Add 7% Outside U.S.A.

THE HACKER'S TREASURE CHEST On Disk

18 Utility Programs on disk. Each program is specifically designed to aid you in backing-up your software collection. You will be able to duplicate **disks, cartridges** and **cassettes.** Any one program is worth the price of all 18. It has taken us over one year to put together this fine collection on the **Hacker's Treasure Chest** disk. Some of the programs you will receive are **Cartridge Copy, Bootspe Maker, Tape to Disk, SectorCopy, The Unprotector, Sector Disassembler, Bad Sector Finder, Modem Progrsm...** plus more. All of these programs plus 10 more on this disk. You will also receive a ***menu*** that will run basic and binary files just by typing the number of the program. Any file on any disk will load automatically from this menu. ALL FOR ONLY

$34.95 Plus $3.50 Shipping. Add 7% outside U.S.A.

DENSITY "DOUBLER"

1050

DOUBLE THE POWER... TRIPLE THE SPEED...

Get TRUE double density, full compatibility with any DOS. Now store twice as much data on each disk. Read and write up to 3X faster in single or double density (Whole Track Buffering). Includes ultra-speed software, simple plug-in PC board. No soldering or cutting required.

$59.95 Plus $3.50 Shipping. Add 7% outside U.S.A.

WRITE-RIGHT

This device will allow you to write to side 2 of any disk. Install this box to your **ATARI™** 1050 in 5 minutes. Just plug in one cable - no cutting or soldering required. Push a button and a LED will light, allowing you to write to a disk without notching out a hole in the disk. Easy plug-in installation. Instructions included. Fully tested & assembled.

$29.95 Plus $3.50 Shipping. Add 7% outside U.S.A.

CART CLONE™

COPY ANY ATARI™ CARTRIDGE

A MUST FOR ALL ATARI™ USERS. **CART CLONE** will backup and transfer any 8 or 16k cartridge to disk or tape. The contents of the cartridge will become a file which you can transfer, rename or delete. They will execute from DOS. No need to run a special menu or program to run these files. It goes in the left cartridge slot enabling it to work in all ATARI™ Home Computers including the XL series. You can get **CART CLONE™** with software for

$59.95 Plus $3.50 Shipping. Add 7% outside U.S.A.

"Constantly Working on New Products and Software."

DUPLICATING TECHNOLOGIES inc.

Formerly Gardner Computing

99 Jericho Tpke., Suite 302A Jericho, N.Y. 11753

TECHNICAL INFO ONLY **(516) 333-5504, 5950**

WEEKDAY ORDERS ONLY **(516) 333-5805, 5807, 5808**

EVE & WKEND ORDERS ONLY **(516) 333-5950**

TERMS *We accept American Express, Visa, MasterCard and C.O.D. orders. Foreign orders must be in U.S. dollars. All personal checks allow 14 days to clear. Most items shipped within 24 hours.*

Classified

SOFTWARE

LOTTO PICKER. Improve your chances for those Million Dollar Jackpots! Picks LOTTO, WIN-4 and Daily Numbers. All USA & CAN. games incl'd. Expandable! IBM/C64/TI99 $29.95. Order Now! 1-800-341-1950 ext. 77. Mail Orders: Ridge, 170 B'way, #201C, NY, NY 10038. Catalog.

FREE SOFTWARE for C64, C128, IBM, & CPM. For info send self-addressed, stamped envelope to: PUBLIC DOMAIN USERS GROUP, PO Box 1442-A1, Orange Park, FL 32067

TI-99/4A QUALITY SOFTWARE for Business, Home and Entertainment ** Bonus Software Offer! ** Send for FREE catalog to MICRO-BIZ HAWAII, BOX 1108, PEARL CITY, HI 96782

COMMODORE: TRY BEFORE YOU BUY. Top 25 best-selling games, utilities, new releases. Visa, MasterCard. Free brochure. Rent-A-Disk, 908 9th Ave., Huntington, WV 25701 (304) 522-1665

DISCOUNT SOFTWARE. Amiga/Apple/Atari/C64-128/IBM PC-PCjr/TRS-80/Timex/Sinclair. Free Catalog: WMJ DATA SYSTEMS, 4 Butterfly Dr., Hauppauge, NY 11788

Genealogy Programs for the 64 & 128. **Family Tree** will produce Pedigree Charts, Family Group Records, Individual Files, Indexes, Searches of Ancestors. LDS version available. $49.95 Genealogy Software, POB 1151, Port Huron, MI 48061, (519) 344-3990

FREE SOFTWARE CATALOG
for APPLE, ATARI, C64, IBM, ST - 1/3 OFF retail prices. We carry SSI, Infocom, Elec. Arts, Microprose, and many more. TEVEX, Dept C, 1710 Wilwat Dr. #E, Norcross, GA 30093

LINCAD, CALCAD, FFT-SPECTRUM ANALYSIS. VALUABLE PROGRAMS FOR ELECTRONIC CIRCUIT OR SIGNAL ANALYSIS & DESIGN LINCAD FOR LINEAR CIR.-C64:$49, IBM-PC:$99. CALCAD FOR LOGIC ANALYSIS + SYNTHESIS - IBM-PC:$179. FFT ANALYZER - C64:$49, IBM-PC:$99. +$3.50 S/H. OH RES. ADD 5.5% TAX. CHECK OR M.O. WRITE FOR DETAILS. SOFCAD ELECTRONICS, INC., PO BOX 21845, COLUMBUS, OH 43221

HEY AMIGO! PD Software for Amiga. Games, graphics, utilities, more! Over 15 disks available only $7.95 each! SASE for list Casaware, PO Box 1646, Orange Park, FL 32067

WIN AT CASINO BLACKJACK! Disk includes game, play analyzer and powerful point count system. Proves itself on your machine. For IBM-PC (no graphics card req'd), Atari 800 (48K)/XL/XE. Specify. $19.95. BISONSOFT, Box 2136, Bloomington, IN 47402

$ LUCKY LOTTO $ — Lotto pgm for all states. Over 2 yr. R&D. For 64K + disk. IBM, CoCoII, C64, Tan.1000, $49.95 + $5 s/h to: Delta, POB 968, Concord, NC 28026 (spec st.)

TANDY 1000 PROGRAMS AND NEWSLETTER Send for free information on educational & entertainment programs & newsletter. Soda Pop Software, POB 653, Kenosha, WI 53141

REAL PINOCHLE. Double-deck, 4-handed, partnership for one player. For C64, PC, PCjr. On disk $20. Jim Bernard, 301 Forest Dr., Bellevue, NE 68005

BRIDGE GAME PROGRAM $39.95 demo disk $5 1 to 4 players for IBM, Apple, TI99, TRS80, C64, 128, +4, 16, VIC. Authors John & Lynda Allan, Box 313, Azilda, Ontario, Canada P0M 1B0

Animal Records maintained with **Petigree** for the C64. Produces Litter, Awards, Breeding, Show, Individual Records, Pedigree Charts. $69.95. Genealogy Software, POB 1151, Port Huron, MI 48061, (519) 344-3990

FREE! PC PUBLIC DOMAIN SOFTWARE! MS-DOS, IBM & Compatibles - Save $$$ @ $2.50 per disk! Free info. AP-JP, Inc., PO Box 1155, W. Babylon, NY 11704

HARDWARE

ASSEMBLE YOUR OWN JOYSTICKS & SAVE $$
Kits available for IBM, Apple, Commodore, Atari, TI99/4A. Schematics, plastics, cables, components included. Send $24.95 for Apple or IBM, $12.95 for TI99/4A, $9.95 for Commodore/Atari JOYSTICK KITS, BOX 8248, LONGVIEW, TX 75607 214-758-0040 Specify Computer type.

REPAIR YOUR OWN JOYSTICKS & SAVE $$
Repair kits available for P.P.I. & TG Product Joysticks. Send $1 and SASE for Parts & Price List. JOYSTICK KITS, BOX 8248, LONGVIEW, TX 75607 214-758-0040

MISCELLANEOUS

IBM PCjr. REPORT: THE NATIONAL NEWS-LETTER PCjr.-specific articles, reviews, Public Domain from across the nation. $18/yr. PCjr CLUB, POB 95067, Schaumburg, IL 60195

MIDWEST COMMODORE CONFERENCE/EXPO C64, C128, AMIGA - AUG. 9 & 10, 1986 - $20/person. Greater Omaha Commodore Users Group. PO Box 24115, Omaha, NE 68124

FLIGHT SIMULATOR II CONTROL CONSOLE Complete set of plans for aircraft-style controls and panel. C64. $15 to: FLIGHT SYSTEMS, Rte 2, Box 121, Gaston, OR 97119

COMPUTE! Classified is a low-cost way to tell over 350,000 microcomputer owners about your product or service.

Rates: $25 per line, minimum of four lines Any or all of the first line set in capital letters at no charge Add $15 per line for boldface words, or $50 for the entire ad set in boldface (any number of lines) Inquire about display rates

Terms: Prepayment is required Check, money order, American Express, Visa, or MasterCard is accepted Make checks payable to COMPUTE! Publications

Form: Ads are subject to publisher's approval and must be either typed or legibly printed One line equals 40 letters and spaces between words Please underline words to be set in boldface

General Information: Advertisers using post office box numbers in their ads must supply permanent address and telephone numbers Orders will not be acknowledged Ad will appear in next available issue after receipt

Closing: 10th of the third month preceding cover date (e g , June issue closes March 10th) Send order and remittance to: Harry Blair, Classified Manager, COMPUTE!, P.O. Box 5406, Greensboro, NC 27403 To place an ad by phone, call Harry Blair at (919) 275-9809

Notice: COMPUTE! Publications cannot be responsible for offers or claims of advertisers, but will attempt to screen out misleading or questionable copy

Advertisers Index

To Our Readers:

COMPUTE! Publications is a part of the ABC Consumer Magazines group of ABC Publishing, Inc. and recently we consolidated many of our operations and moved our Customer Service Department to the New York ABC headquarters. If you have any questions regarding back issues, disk orders, book orders, or how to place an order, call toll free **1-800-346-6767**. New York residents should call 212-887-8525.

If you want to order a subscription to COMPUTE!, COMPUTE!'s GAZETTE, COMPUTE!'s GAZETTE DISK, or the COMPUTE! DISK, call **1-800-247-5470** or in Iowa call 1-800-532-1272.

Our Editorial Offices remain in Greensboro, North Carolina. If you wish to submit an article for publication, write us at COMPUTE! Publications, Inc., P.O. Box 5406, Greensboro, NC 27403.

We thank you for your interest and continued support of COMPUTE! Publications.

COMPUTE! Publications, Inc. abc
Part of ABC Consumer Magazines Inc
One of the ABC Publishing Companies

Flight Simulator II Scenery Disks

The Challenge of Accomplished Flight

With a realism comparable to (and in some ways even surpassing) $100,000 aircraft flight simulators, Flight Simulator II includes full flight instrumentation and avionics, and provides a full-color out-the-window view. Instruments are arranged in the format standard to modern aircraft. All the radios needed for IFR flight are included. Front, rear, left, right, and diagonal views let you look in any direction. Program features are clearly documented in a 96-page Pilot's Operating Handbook.

For training in proper flight techniques, Flight Simulator II includes another 96-page instruction manual, compiled by two professional flight instructors with over 8,000 hours flight time and 12,000 hours of aviation teaching experience. You'll learn correct FAA-recommended flight procedures, from basic aircraft control through instrument approaches. To reward your accomplishments, the manual even includes a section on aerobatic maneuvers.

The Realism and Beauty of Flight

Go sight-seeing over detailed, realistic United States scenery. High-speed graphic drivers provide an animated out-the-window view in either day, dusk, or night flying modes.

Flight Simulator II features over 80 airports in four different scenery areas: New York, Chicago, Seattle, and Los Angeles. Six additional Scenery Disks covering the entire Western half of the United States are now available in IBM and C64/128 disk formats.

Apple and Atari versions will be released soon. Each disk covers a geographical region of the country in detail, and is very reasonably priced.

The Pure Fun of "World War I Ace"

When you think you're ready, you can test your flying skills with the "World War I Ace" aerial battle game. This game sends you on a bombing run over heavily-defended enemy territory. Six enemy fighters will attempt to engage you in combat as soon as war is declared. Your aircraft can carry five bombs, and your machine guns are loaded with 100 rounds of ammunition.

See Your Dealer. Flight Simulator II is available on disk for the Apple II, Atari XL/XE, and Commodore 64/128 computers for $49.95. Scenery Disks for the C64 and IBM PC (Jet or Microsoft Flight Simulator) are $19.95 each. A complete Western U.S. Scenery six-disk set is also available for $99.95. For additional product or ordering information, call (800) 637-4983.

subLOGIC Corporation
713 Edgebrook Drive
Champaign IL 61820
(217) 359-8482 Telex: 206995

Order Line: (800) 637-4983
(except in Illinois, Alaska, and Hawaii)

BATTERIES INCLUDED

Atari ST Software

Integral Solutions

D·E·G·A·S

DESIGN & ENTERTAINMENT GRAPHIC-ARTS SYSTEM

by Tom Hudson

The artistic standard for the ST! Beautiful graphics program for business and pleasure. All the key drawing/painting functions, text integration, and graphic design tools! Available: Now!

TIME LINK

by Softechnics

Scheduling & Time-keeping tool for home and business. Your day, week, month, year at a glance. Many incredible uses! Available: Now!

THUNDER!

by Mark Skapinker

Use this unique real time spelling checker desk accessory within **any** *ST GEM application. 50,000 word real time spelling checker. Abbreviations function completes a word when you enter the first letters. Amazing speed. Available: 3rd Quarter 1986.*

HomePak

by Russ Wetmore

ST version of InfoWorld's Best Buy of the Year Award! 3 integrated programs on one disk. Telecommunications. Word-processor. Information manager. The easy answer to three key software needs. Available: 3rd Quarter 1986.

BTS THE SPREADSHEET

by Alan Porter, Martin Rezmer and Jason Loveman

Two in one! *Sophisticated full featured spreadsheet program. All the key Math, Stats, Logical and Financial functions. 1000 x 1000 worksheet.* **Plus** *Desk accessory version on same disk! Available: 3rd Quarter 1986.*

The Consultant

by the B.I. Software Development Team

THE ultimate relational data base. Easy to learn. Unique new features. Power and sophistication you can use right away. For business or personal use. Available: 4th Quarter 1986.

THE I·SGUR Portfolio SYSTEM

by Lee Isgur and Mark Skapinker

Investment management program designed for private investors and professionals. On-line portfolio updating. Analytical functions for more profitable decisions. A PC Magazine editor's choice! Available: Now!

I·S TALK

by Stephen Couchman

Full-scale telecommunications program. Easy to use and virtually error-proof. Includes 50,000 word spelling checker and three levels of macros. Available: Now!

PaperClip Elite

by Dan Moore and Steve Ahlstrom

Next generation word-processor. All the high-productivity features plus a real-time spell checker, idea processing, integrated text/graphics, much more! Available: 3rd Quarter 1986.

D·E·G·A·S Elite

DESIGN & ENTERTAINMENT GRAPHIC ARTS SYSTEM

by Tom Hudson

Second-generation graphics program creates presentation-quality visuals. Full range of design/color functions. Multiple workscreens, new fonts, many other enhancements. The deluxe Degas Elite is totally compatible with all other Degas files! Integrate your Degas Elite pictures with PaperClip Elite text files. Available: 3rd Quarter 1986.

B/GRAPH™ Elite

by Alan Page, Joe Chiazzese and Robert Wilson

Serious graphics/charting and statistics desk package. Pie charts, 2 and 3 dimensional bar charts, area graphs, much more. Change designs without re-entering data. Make beautiful presentations. Available: 4th Quarter 1986.

I·S TIME AND BILLING

by Roy Miller

Office management program for professionals. Functions include Daily Records, Automatic Billing, Accounts Receivable, Billing Breakdowns and more! Available: 4th Quarter 1986.

I·S TALK, TIME & BILLING, PAPERCLIP ELITE, CONSULTANT, DEGAS ELITE, DEGAS, PORTFOLIO, BTS SPREADSHEET, HOMEPAK, THUNDER AND B/GRAPH ARE ALL FILE COMPATIBLE, OF COURSE!

BATTERIES INCLUDED

WRITE TO US FOR FULL COLOUR CATALOG OF OUR PRODUCTS FOR COMMODORE, ATARI, APPLE AND IBM SYSTEMS. FOR TECHNICAL SUPPORT OR PRODUCT INFORMATION PLEASE PHONE (416) 881-9816. 30 MURAL STREET, RICHMOND HILL, ONTARIO L4B 1B5 CANADA, (416) 881-9941, TELEX: 06-986-266. 17875 SKY PARK NORTH, SUITE P, IRVINE, CALIFORNIA, USA 92714, (416) 881-9816, Telex: 509-139.

I·S Integral Solutions